Java学习群

72030155

进群可以免费获取视频教程以及每日免费听老师讲课

前　言

本书的目的在于快速提供使用T-SQL语言解决问题和执行任务的技巧。为了让你快速了解各个任务及其相关的T-SQL解决方案，我以问题/解决方案的形式编写了本书。你可以从中查找你希望执行的任务，看一下怎么做，然后在你自己的系统中执行。本书编写时遵循了以下几条关键原则。

- 保持简洁，仅提供完成工作所需要的信息。
- 让各种技巧之间和各章之间彼此独立——尽量避免交叉引用和内容分散。
- 主要关注可完全使用T-SQL执行操作的特性。例如，书中会讨论新的资源调控器特性，因为DBA一般会使用T-SQL对它进行部署，但是不会讨论基于策略的管理，因为它对SQL Server代理、SQL Server Management Objects（SMO）和SQL Server Management Studio有底层依赖。幸运的是，大多数新的SQL Server引擎改进都是完全基于T-SQL的，因此本书会讨论这些内容。
- 涵盖了从初级到高级的各种技巧。每一章都是从基本的技巧开始，然后逐渐上升到更高级的主题。

SQL Server 2008的新特性会贯穿在本书应用它们的各章节中。如果你只是想查找T-SQL更新了哪些新特性，也不用担心，对于每章涉及的新特性，我都会在章首处列出。

尽管本书的关键原则是保持内容简洁，但你会注意到本书还是很厚。这是SQL Server特性集持续扩展的结果，然而请放心，技巧部分还是比较简洁的，并且本书的编排让你能够快速找到完成工作所需要的答案。

本书适合SQL Server开发人员、管理员、应用程序开发人员以及那些负责开发数据库或管理SQL Server环境的IT多面手阅读。你可以从头到尾阅读本书，也可以直接翻到你感兴趣的主题来读。你可以在应聘或考试之前使用本书重温一下某些主题。对于有经验的SQL Server专业人士来说，如果有些命令或技术忘记了，本书可以帮助他们迅速拾起它们的用法。

感谢你的阅读！

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

目　录

第 1 章　SELECT ····· 1

1.1　基本的SELECT语句 ····· 1
1.1.1　从表中选择指定列 ····· 2
1.1.2　从所有行中选择所有列 ····· 2
1.2　使用基本WHERE子句进行有选择的查询 ····· 3
1.2.1　使用WHERE子句指定结果集中返回的行 ····· 3
1.2.2　组合搜索条件 ····· 4
1.2.3　否定搜索条件 ····· 5
1.2.4　保持WHERE子句无歧义 ····· 5
1.3　使用运算符和表达式 ····· 6
1.3.1　在日期范围搜索中使用BETWEEN ····· 7
1.3.2　使用比较运算符 ····· 8
1.3.3　检测NULL值 ····· 9
1.3.4　基于一组值返回行 ····· 9
1.3.5　LIKE和通配符结合使用 ····· 9
1.3.6　声明变量及为变量赋值 ····· 11
1.4　数据分组 ····· 12
1.4.1　使用GROUP BY子句 ····· 12
1.4.2　使用GROUP BY ALL ····· 13
1.4.3　使用HAVING选择性地查询分组的数据 ····· 13
1.5　对结果排序 ····· 14
1.5.1　使用ORDER BY子句 ····· 15
1.5.2　在排序的结果中使用TOP关键字 ····· 16
1.6　SELECT子句技术 ····· 18
1.6.1　使用DISTINCT消除重复值 ····· 18
1.6.2　在聚合函数中使用DISTINCT ····· 18
1.6.3　使用列别名 ····· 19
1.6.4　使用SELECT创建脚本 ····· 20
1.6.5　字符串拼接 ····· 21
1.6.6　使用SELECT创建逗号分隔的列表 ····· 21
1.6.7　使用INTO子句 ····· 22
1.7　子查询 ····· 23
1.8　从多个数据源查询 ····· 24
1.8.1　使用内联结 ····· 25
1.8.2　使用外联结 ····· 26
1.8.3　使用交叉联结 ····· 27
1.8.4　在同一查询中多次引用同一个表 ····· 27
1.8.5　使用衍生表 ····· 28
1.8.6　使用UNION组合结果集 ····· 29
1.9　使用APPLY来为每行调用表值函数 ····· 30
1.9.1　使用CROSS APPLY ····· 30
1.9.2　使用OUTER APPLY ····· 32
1.10　数据源高级技术 ····· 33
1.10.1　使用TABLESAMPLE来返回随机行 ····· 33
1.10.2　使用PIVOT把单列值转化为多列和聚合数据 ····· 34
1.10.3　使用UNPIVOT规范化数据 ····· 36
1.10.4　使用EXCEPT和INTERSECT返回不重复的或匹配的行 ····· 38
1.11　汇总数据 ····· 40
1.11.1　使用CUBE汇总数据 ····· 40
1.11.2　使用ROLLUP来汇总数据 ····· 42
1.11.3　使用分组集创建自定义汇总 ····· 43
1.11.4　展现GROUPING生成的行 ····· 44

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

1.11.5 使用GROUPING_ID标识分组级别 ····· 46
1.12 公共表表达式 ····· 49
1.12.1 使用非递归的公共表表达式 ····· 49
1.12.2 使用递归的公共表表达式 ····· 52
第2章 执行、捕获和跟踪数据修改 ····· 55
2.1 INSERT ····· 55
2.1.1 向表中插入一行 ····· 55
2.1.2 使用默认值插入行 ····· 56
2.1.3 显式向一个IDENTITY列插入值 ····· 57
2.1.4 在表中插入拥有uniqueidentifier列的行 ····· 59
2.1.5 使用INSERT...SELECT语句插入行 ····· 59
2.1.6 调用存储过程插入数据 ····· 60
2.1.7 通过VALUES插入多个行 ····· 62
2.1.8 将VALUES作为表源来使用 ····· 62
2.2 UPDATE ····· 63
2.2.1 更新一行 ····· 64
2.2.2 根据FROM和WHERE子句更新行 ····· 65
2.2.3 更新大值数据类型的列 ····· 66
2.2.4 使用OPENROWSET和BULK插入或更新图片文件 ····· 68
2.2.5 在维护SQL Server事务控制时向文件系统存储非结构化数据 ····· 69
2.2.6 “就地”赋值并修改数据库值 ····· 73
2.3 DELETE ····· 75
2.3.1 删除行 ····· 75
2.3.2 截断表 ····· 76
2.4 高级数据修改技术 ····· 77
2.4.1 使用TOP分块修改数据 ····· 77
2.4.2 在一条语句中执行INSERT、UPDATE和DELETE ····· 78
2.5 捕获和跟踪数据修改的改变 ····· 81
2.5.1 返回数据修改语句影响的行 ····· 81
2.5.2 异步捕获表数据修改 ····· 83
2.5.3 从CDC表查询所有变更 ····· 87
2.5.4 从CDC表查询净变更 ····· 89
2.5.5 翻译CDC更新掩码 ····· 90
2.5.6 使用LSN边界 ····· 92
2.5.7 禁用对表和数据库的变更数据捕获 ····· 93
2.5.8 以最小的磁盘开销跟踪净数据更改 ····· 93
第3章 事务、锁定、阻塞和死锁 ····· 100
3.1 事务控制 ····· 100
3.1.1 使用显式事务 ····· 101
3.1.2 使用DBCC OPENTRAN显示最早的活动事务 ····· 104
3.1.3 通过会话查询事务信息 ····· 104
3.2 锁定 ····· 106
3.2.1 查看锁的活动 ····· 108
3.2.2 控制表的锁升级行为 ····· 109
3.3 事务、锁定和并发 ····· 110
3.4 阻塞 ····· 115
3.4.1 找到并解决阻塞进程 ····· 116
3.4.2 配置语句等待锁释放的时长 ····· 118
3.5 死锁 ····· 118
3.5.1 使用跟踪标志位找出死锁 ····· 119
3.5.2 设置死锁优先级 ····· 121
第4章 表 ····· 123
4.1 表基础 ····· 123
4.1.1 创建表 ····· 126
4.1.2 为既有表添加列 ····· 126
4.1.3 修改列定义 ····· 127
4.1.4 创建计算列 ····· 128
4.1.5 减少NULL列的存储空间 ····· 129
4.1.6 删除表中的列 ····· 131
4.1.7 报告表信息 ····· 132
4.1.8 删除表 ····· 132
4.2 排序规则基础 ····· 133
4.2.1 查看排序规则元数据 ····· 133
4.2.2 指定列的排序规则 ····· 134
4.3 键 ····· 134
4.3.1 创建有主键的表 ····· 135
4.3.2 为既有表增加主键约束 ····· 136
4.3.3 创建一个有外键引用的表 ····· 136

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

4.3.4 为既有表增加外键 …… 137
4.3.5 创建递归外键引用 …… 138
4.3.6 允许外键的级联修改 …… 139
4.4 代理键 …… 140
4.4.1 在创建表的时候使用IDENTITY属性 …… 141
4.4.2 使用DBCC CHECKIDENT来查看和纠正IDENTITY种子值 …… 142
4.4.3 使用ROWGUIDCOL属性 …… 143
4.5 约束 …… 143
4.5.1 创建唯一约束 …… 144
4.5.2 为既有表增加UNIQUE约束 …… 145
4.5.3 使用CHECK约束 …… 145
4.5.4 为既有表增加CHECK约束 …… 146
4.5.5 禁止和启用约束 …… 147
4.5.6 在表创建时使用DEFAULT约束 …… 148
4.5.7 为既有表增加DEFAULT约束 …… 149
4.5.8 从表删除约束 …… 150
4.6 临时表和表变量 …… 150
4.6.1 使用临时表在批处理中进行多次查找 …… 151
4.6.2 创建表变量来保存临时结果集 …… 152
4.7 管理超大型表 …… 153
4.7.1 实现表分区 …… 154
4.7.2 确定数据在分区中的位置 …… 157
4.7.3 增加新的分区 …… 158
4.7.4 移除分区 …… 160
4.7.5 把分区移动到不同的表 …… 161
4.7.6 移除分区函数和分区方案 …… 162
4.7.7 用文件组提高VLDB的可管理性 …… 162
4.7.8 用数据压缩减少磁盘空间使用 …… 164

第 5 章 索引 …… 167

5.1 索引预览 …… 167
5.1.1 创建表索引 …… 169
5.1.2 在非键列上强制唯一性 …… 170
5.1.3 在多个列上创建索引 …… 171
5.1.4 定义索引列排序方向 …… 172
5.1.5 查看索引元数据 …… 172
5.1.6 禁用索引 …… 174
5.1.7 删除索引 …… 174
5.1.8 使用DROP_EXISTING改变既有索引 …… 174
5.2 控制索引创建的性能和并发性 …… 175
5.2.1 在tempdb中创建临时索引 …… 175
5.2.2 控制索引创建的并行执行计划 …… 176
5.2.3 在索引创建过程中允许用户表访问 …… 176
5.3 索引选项 …… 177
5.3.1 使用索引INCLUDE …… 177
5.3.2 使用PAD_INDEX和FILLFACTOR …… 178
5.3.3 禁用页和/或行索引锁定 …… 178
5.4 管理超大型索引 …… 179
5.4.1 在文件组上创建索引 …… 180
5.4.2 实现索引分区 …… 180
5.4.3 为行的子集做索引 …… 181
5.4.4 减小索引的大小 …… 182

第 6 章 全文搜索 …… 183

6.1 全文索引和全文目录 …… 183
6.1.1 创建全文目录 …… 183
6.1.2 创建全文索引 …… 184
6.1.3 修改全文目录 …… 186
6.1.4 修改全文索引 …… 187
6.1.5 检索全文目录和索引元数据 …… 189
6.1.6 从全文索引中去掉普通字符串 …… 190
6.1.7 删除全文索引 …… 193
6.1.8 删除全文目录 …… 193
6.2 基本搜索 …… 194
6.2.1 使用FREETEXT来搜索全文索引的列 …… 194
6.2.2 使用CONTAINS来搜索单词 …… 195
6.3 高级搜索 …… 196
6.3.1 使用CONTAINS和通配符来搜索 …… 196
6.3.2 使用CONTAINS来搜索变形匹配 …… 196
6.3.3 使用CONTAINS根据词的相邻搜索结果 …… 197
6.4 排名搜索 …… 197
6.4.1 根据含义返回排名搜索结果 …… 198

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

6.4.2 根据权值返回排名搜索结果……199
第7章 视图……201
7.1 普通视图……201
7.1.1 创建基本视图……202
7.1.2 查询视图定义……203
7.1.3 显示视图及其结构……204
7.1.4 刷新视图定义……205
7.1.5 修改视图……206
7.1.6 删除视图……206
7.1.7 使用视图修改数据……206
7.2 视图加密……207
7.3 索引视图……208
7.3.1 创建索引视图……208
7.3.2 强制优化器为索引视图使用索引……210
7.4 分区视图……211
第8章 SQL Server 函数……216
8.1 聚合函数……216
8.1.1 返回平均值……217
8.1.2 返回行总数……217
8.1.3 找出表达式中的最小值和最大值……218
8.1.4 返回值的和……218
8.1.5 使用统计聚合函数……219
8.2 数学函数……219
8.3 字符串函数……221
8.3.1 把字符值转化为ASCII以及把ASCII转回字符……222
8.3.2 返回整数和字符Unicode值……222
8.3.3 获取某个字符串在另一个字符串中的起始位置……223
8.3.4 使用通配符找到某个字符串在另一个字符串中的起始位置……223
8.3.5 检测字符串相似度……224
8.3.6 获取字符串最左和最右部分……224
8.3.7 检测字符串中的字符数或者字节数……225
8.3.8 把字符串的一部分替换成另一个字符串……226
8.3.9 把字符串填充到字符串中……226
8.3.10 在小写和大写之间进行转化……227
8.3.11 移除前导空格和尾部空格……228
8.3.12 重复一个表达式N次……228
8.3.13 重复一个空格N次……229
8.3.14 逆序输出表达式……229
8.3.15 返回表达式的一块……229
8.4 处理NULL……230
8.4.1 用另一个值替换NULL值……230
8.4.2 使用ISNULL进行灵活的搜索……230
8.4.3 返回表达式列表中第一个非NULL值……232
8.4.4 当两个表达式相等的时候返回NULL值，否则返回第一个表达式……232
8.5 日期函数……233
8.5.1 返回当前日期和时间……233
8.5.2 在时区间进行转换……234
8.5.3 增加或减少日期值……235
8.5.4 找出两个日期的差……236
8.5.5 显示日期一部分的字符串值……237
8.5.6 使用DATEPART显示日期一部分的整数值……237
8.5.7 使用YEAR、MONTH和DAY显示日期一部分的整数值……238
8.6 类型转换……238
8.6.1 转换数据类型……238
8.6.2 将日期转换为文本形式……239
8.6.3 以字符串字面量表示二进制数据……240
8.6.4 计算表达式返回的数据类型……241
8.7 排名函数……242
8.7.1 生成递增行号……242
8.7.2 根据排名返回行……244
8.7.3 根据无间隔排名返回行……245
8.7.4 使用NTILE……246
8.8 使用系统函数检测服务器、数据库以及连接级别的配置……246
8.8.1 确定每周的第一天……246
8.8.2 查看当前会话使用的语言……247

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

8.8.3 查看和设置当前连接锁超时设置……247
8.8.4 显示当前存储过程上下文的嵌套级别……248
8.8.5 返回当前SQL Server实例名和SQL Server版本……248
8.8.6 返回当前连接的会话ID（SPID）……249
8.8.7 返回打开事务的数量……249
8.8.8 获取之前语句影响的行数……250
8.8.9 获取系统统计信息……250
8.8.10 显示数据库和SQL Server设置……251
8.8.11 返回当前数据库ID和名称……252
8.8.12 返回数据库对象名和ID……253
8.8.13 返回当前用户会话的应用程序和主机……253
8.8.14 获取当前用户和登录名上下文……253
8.8.15 查看用户连接选项……254
8.9 IDENTITY和uniqueidentifier函数……255
8.9.1 返回最后一个标识值……255
8.9.2 返回标识列的种子和增量值……256
8.9.3 创建新的uniqueidentifier值……256
第 9 章 条件处理、流控制和游标……258
9.1 条件处理……258
9.1.1 使用CASE来计算单个输入表达式……258
9.1.2 使用CASE来计算布尔表达式……260
9.1.3 使用IF...ELSE……261
9.2 流控制……262
9.2.1 使用RETURN……263
9.2.2 使用WHILE……264
9.2.3 使用GOTO……266
9.2.4 使用WAITFOR……267
9.3 游标……268
第 10 章 存储过程……273
10.1 存储过程基础……273
10.1.1 创建基本的存储过程……274
10.1.2 创建带参数的存储过程……275
10.1.3 使用OUTPUT参数……277
10.1.4 修改存储过程……278
10.1.5 删除存储过程……279
10.1.6 在SQL Server启动时自动执行存储过程……279
10.1.7 报告存储过程元数据……281
10.1.8 为存储过程建立文档……281
10.2 存储过程安全……282
10.2.1 加密存储过程……282
10.2.2 使用EXECUTE AS来指定过程的安全上下文……283
10.3 重新编译和缓存……286
10.3.1 每次存储过程执行时重新编译……286
10.3.2 清空过程缓存……287
第 11 章 用户定义函数和类型……289
11.1 UDF基础……289
11.1.1 创建标量用户定义函数……290
11.1.2 创建内联用户定义函数……294
11.1.3 创建多语句用户定义函数……296
11.1.4 修改用户定义函数……299
11.1.5 查看UDF元数据……300
11.1.6 移除用户定义函数……301
11.2 从UDF获益……301
11.2.1 维护可重用的代码……302
11.2.2 交叉引用的自然键值……303
11.2.3 使用多语句UDF替代视图……306
11.3 UDT基础知识……308
11.3.1 创建和使用用户定义类型……308
11.3.2 找出使用用户定义类型的列和参数……310
11.3.3 删除用户定义类型……311
11.3.4 传入表值参数……312
第 12 章 触发器……315
12.1 DML触发器……316
12.1.1 创建AFTER DML触发器……316
12.1.2 创建INSTEAD OF DML触发器……319
12.1.3 在DML触发器中处理事务……322

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

12.1.4 根据修改的列控制DML触发器 ····· 324
12.1.5 查看DML触发器元数据 ····· 325
12.2 DDL触发器 ····· 326
12.2.1 创建审核数据库级别的事件的DDL触发器 ····· 326
12.2.2 创建审核服务器级别的事件的DDL触发器 ····· 328
12.2.3 使用登录触发器 ····· 329
12.2.4 查看DDL触发器元数据 ····· 331
12.3 管理触发器 ····· 332
12.3.1 修改触发器 ····· 332
12.3.2 启用和禁止表触发器 ····· 333
12.3.3 限制触发器嵌套 ····· 334
12.3.4 控制触发器递归 ····· 335
12.3.5 设置触发器触发次序 ····· 336
12.3.6 删除触发器 ····· 338
第 13 章 CLR 集成 ····· 339
13.1 CLR概述 ····· 339
13.2 什么时候使用（不使用）程序集 ····· 340
13.3 CLR对象概述 ····· 341
13.4 创建CLR数据库对象 ····· 341
13.4.1 启用SQL Server的CLR支持 ····· 342
13.4.2 为CLR存储过程写程序集 ····· 342
13.4.3 把程序集编译成DLL文件 ····· 345
13.4.4 把程序集载入SQL Server ····· 345
13.4.5 创建CLR存储过程 ····· 346
13.4.6 创建CLR标量用户定义函数 ····· 348
13.4.7 创建CLR触发器 ····· 350
13.5 管理程序集 ····· 352
13.5.1 查看程序集元数据 ····· 352
13.5.2 修改程序集权限 ····· 352
13.5.3 从数据库移除程序集 ····· 353
第 14 章 XML、分层和空间数据 ····· 354
14.1 使用原生XML ····· 354
14.1.1 创建XML数据类型列 ····· 354
14.1.2 插入XML数据到列 ····· 355
14.1.3 使用架构验证XML数据 ····· 356
14.1.4 获取XML数据 ····· 358
14.1.5 修改XML数据 ····· 361
14.1.6 XML索引 ····· 362
14.2 在XML文档与关系型数据之间进行转换 ····· 363
14.2.1 将关系型数据格式化为XML ····· 363
14.2.2 将XML转换为关系型 ····· 366
14.3 使用原生分层数据 ····· 368
14.3.1 存储分层数据 ····· 368
14.3.2 返回指定的祖先 ····· 370
14.3.3 返回子节点 ····· 371
14.3.4 返回节点的深度 ····· 371
14.3.5 返回根节点 ····· 372
14.3.6 确定节点是否为当前节点的子节点 ····· 372
14.3.7 更改节点位置 ····· 372
14.4 原生空间数据 ····· 373
14.4.1 存储空间数据 ····· 373
14.4.2 查询空间数据 ····· 376
第 15 章 提示 ····· 380
15.1 使用联结提示 ····· 380
15.2 使用查询提示 ····· 382
15.3 使用表提示 ····· 384
15.3.1 不锁定执行查询 ····· 385
15.3.2 强制SEEK覆盖SCAN ····· 385
第 16 章 错误处理 ····· 388
16.1 系统定义和用户定义的错误消息 ····· 388
16.1.1 查看系统错误信息 ····· 388
16.1.2 创建用户定义错误消息 ····· 389
16.1.3 删除用户定义错误消息 ····· 391
16.2 手动引发错误 ····· 391
16.3 捕获和处理应用程序错误 ····· 393
16.3.1 旧风格的错误处理 ····· 394
16.3.2 使用TRY...CATCH进行错误处理 ····· 396
16.3.3 不重编写存储过程来应用TRY...CATCH ····· 398
16.3.4 嵌套错误处理 ····· 398

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 17 章 主体 ····· 401
17.1 Windows级别的主体 ····· 401
17.1.1 创建Windows登录名 ····· 402
17.1.2 查看Windows登录名 ····· 403
17.1.3 修改Windows登录名 ····· 403
17.1.4 删除Windows登录名 ····· 404
17.1.5 拒绝Windows用户或用户组的SQL Server访问 ····· 405
17.2 SQL Server级别的主体 ····· 405
17.2.1 创建SQL Server登录名 ····· 407
17.2.2 查看SQL Server登录名 ····· 407
17.2.3 修改SQL Server登录名 ····· 408
17.2.4 管理登录名的密码 ····· 409
17.2.5 删除SQL登录名 ····· 409
17.2.6 管理服务器角色成员 ····· 410
17.2.7 报告固定服务器角色信息 ····· 411
17.3 数据库级别的主体 ····· 412
17.3.1 创建数据库用户 ····· 413
17.3.2 报告数据库用户信息 ····· 414
17.3.3 修改数据库用户 ····· 414
17.3.4 从数据库删除数据库用户 ····· 415
17.3.5 修复孤立的数据库用户 ····· 415
17.3.6 报告固定数据库角色信息 ····· 416
17.3.7 管理固定数据库角色成员 ····· 417
17.3.8 管理用户定义的数据库角色 ····· 418
17.3.9 管理应用程序角色 ····· 420
第 18 章 安全对象、权限和审核 ····· 423
18.1 权限总览 ····· 424
18.2 服务器范围的安全对象和权限 ····· 426
18.2.1 管理服务器权限 ····· 428
18.2.2 查询服务器权限 ····· 429
18.3 数据库范围的安全对象和权限 ····· 430
18.3.1 管理数据库权限 ····· 431
18.3.2 查询数据库权限 ····· 432
18.4 架构范围的安全对象和权限 ····· 434
18.4.1 管理架构 ····· 436
18.4.2 管理架构的权限 ····· 437
18.5 对象的权限 ····· 438
18.6 管理权限访问安全对象范围 ····· 440
18.6.1 检测当前连接的安全对象的权限 ····· 440
18.6.2 通过安全对象范围报告主体的权限 ····· 441
18.6.3 改变安全对象所有权 ····· 445
18.6.4 允许SQL登录名访问非SQL Server的资源 ····· 446
18.7 对安全对象的主体审核SQL实例和数据库级别活动 ····· 447
18.7.1 定义审核数据源 ····· 447
18.7.2 捕获SQL实例范围的事件 ····· 450
18.7.3 捕获数据库范围的事件 ····· 452
18.7.4 查询捕获的审核数据 ····· 455
18.7.5 管理、修改和移除审核对象 ····· 458
第 19 章 加密 ····· 462
19.1 通过通行短语加密 ····· 462
19.2 主密钥 ····· 464
19.2.1 备份及还原服务主密钥 ····· 465
19.2.2 创建、重新生成以及删除数据库主密钥 ····· 466
19.2.3 备份及还原数据库主密钥 ····· 467
19.2.4 从数据库主密钥删除服务主密钥加密 ····· 468
19.3 非对称密钥加密 ····· 469
19.3.1 创建非对称密钥 ····· 469
19.3.2 查看当前数据库中的非对称密钥 ····· 470
19.3.3 修改非对称密钥的私钥密码 ····· 470
19.3.4 使用非对称密钥对数据进行加密和解密 ····· 471
19.3.5 删除非对称密钥 ····· 473
19.4 对称密钥加密 ····· 473
19.4.1 创建对称密钥 ····· 474
19.4.2 查看当前数据库中的对称密钥 ····· 475
19.4.3 改变对称密钥加密方式 ····· 475
19.4.4 使用对称密钥进行加密和解密 ····· 476
19.4.5 删除对称密钥 ····· 479

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

19.5 证书加密 …… 479
19.5.1 创建数据库证书 …… 479
19.5.2 查看数据库中的证书 …… 480
19.5.3 备份和还原证书 …… 480
19.5.4 管理证书的私钥 …… 482
19.5.5 使用证书加密和解密 …… 483
19.5.6 通过对称密钥自动打开和解密 …… 484
19.6 透明数据加密 …… 486
19.6.1 启用透明数据加密 …… 486
19.6.2 管理和移除TDE …… 487
第 20 章 Service Broker …… 489
20.1 示例场景：在线书店 …… 489
20.2 创建基本的Service Broker应用程序 …… 490
20.2.1 启用数据库的Service Broker活动 …… 491
20.2.2 创建加密用的数据库主密钥 …… 491
20.2.3 管理消息类型 …… 492
20.2.4 创建约定 …… 493
20.2.5 创建队列 …… 494
20.2.6 创建服务 …… 496
20.2.7 启动对话 …… 497
20.2.8 查询队列中传入的消息 …… 499
20.2.9 检索并响应消息 …… 499
20.2.10 结束会话 …… 501
20.2.11 设定Service Broker会话的优先级 …… 503
20.3 创建用来处理消息的存储过程 …… 505
20.4 远程服务器Service Broker的实现 …… 507
20.4.1 启用传输安全模式 …… 510
20.4.2 启用对话安全模式 …… 512
20.4.3 创建路由和远程服务绑定 …… 514
20.5 事件通知 …… 517
第 21 章 配置和查看 SQL Server 的选项 …… 520
第 22 章 创建和配置数据库 …… 524
22.1 创建、修改和删除数据库 …… 524
22.1.1 使用默认配置创建数据库 …… 524
22.1.2 查看数据库信息 …… 525
22.1.3 使用文件选项创建数据库 …… 526
22.1.4 使用用户定义文件组创建数据库 …… 528
22.1.5 设置数据库用户访问 …… 530
22.1.6 重命名数据库 …… 532
22.1.7 删除数据库 …… 533
22.1.8 分离数据库 …… 533
22.1.9 附加数据库 …… 535
22.2 配置数据库选项 …… 535
22.2.1 查看数据库选项 …… 536
22.2.2 配置ANSI SQL选项 …… 536
22.2.3 配置自动选项 …… 538
22.2.4 创建或修改允许外部访问的数据库 …… 539
22.2.5 使用非服务器默认排序规则创建或改变数据库 …… 540
22.2.6 配置游标选项 …… 541
22.2.7 启用日期相关性优化 …… 542
22.2.8 修改数据库参数化行为 …… 543
22.2.9 为事务启用读取一致性 …… 546
22.2.10 配置数据库恢复模式 …… 547
22.2.11 配置页验证 …… 548
22.3 控制数据库访问和拥有关系 …… 550
22.3.1 修改数据库状态为联机、离线或紧急状态 …… 550
22.3.2 修改数据库拥有者 …… 551
22.4 管理数据库文件和文件组 …… 552
22.4.1 为既有的数据库添加数据文件或日志文件 …… 552
22.4.2 从数据库删除数据或日志文件 …… 553
22.4.3 重新分配数据或事务日志文件 …… 554
22.4.4 修改文件的逻辑名称 …… 555
22.4.5 增加数据库文件的大小和修改它的增长选项 …… 556
22.4.6 为既有的数据库添加文件组 …… 557
22.4.7 设置默认文件组 …… 557
22.4.8 删除文件组 …… 558

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

22.4.9 使数据库或文件组为只读……559
22.5 查看和管理数据库空间使用……560
22.5.1 查看数据库空间使用情况……560
22.5.2 收缩数据库或数据库文件……561
第 23 章 数据库完整性和优化……565
23.1 数据库检查……565
23.1.1 使用DBCC CHECKALLOC检查磁盘空间分配结构的一致性…565
23.1.2 使用DBCC CHECKDB检查所有数据库的分配和结构完整性…567
23.2 表和约束……569
23.2.1 使用DBCC CHECKFILEGROUP检查文件组中所有表的分配和结构完整性……569
23.2.2 使用DBCC CHECKTABLE检查表和索引视图的数据完整性……570
23.2.3 使用DBCC CHECKCONSTRAINTS检查表的完整性……573
23.2.4 使用DBCC CHECKCATALOG检查系统表的一致性……574
23.3 索引维护……575
23.3.1 重建索引……575
23.3.2 索引碎片整理……577
23.3.3 重建堆……578
第 24 章 维护数据库对象和对象依赖关系……579
24.1 数据库对象维护……579
24.1.1 修改用户创建的数据库对象的名称……579
24.1.2 修改对象的架构……581
24.2 对象依赖关系……581
24.2.1 标识对象依赖关系……581
24.2.2 标识引用和被引用的实体……583
24.2.3 查看对象定义……585
第 25 章 数据库镜像……587
25.1 上下文中的数据库镜像……587
25.2 数据库镜像体系结构……588
25.3 设置数据库镜像……589
25.3.1 创建镜像端点……590
25.3.2 备份和还原主体数据库……593
25.3.3 创建数据库镜像会话……595
25.4 设置总结……597
25.5 运行数据库镜像……598
25.5.1 改变运行模式……598
25.5.2 进行故障转移……599
25.5.3 暂停或继续镜像会话……600
25.5.4 停止镜像会话和删除端点……600
25.6 镜像和配置选项……601
25.6.1 监视镜像状态……601
25.6.2 配置连接超时时限……601
第 26 章 数据库快照……603
26.1 快照基础……603
26.1.1 创建和查询数据库快照……603
26.1.2 删除数据库快照……605
26.1.3 从数据库快照恢复数据……605
第 27 章 链接服务器和分布式查询……608
27.1 链接服务器基础……608
27.1.1 为另一SQL Server实例创建链接服务器……609
27.1.2 配置链接服务器属性……610
27.1.3 查看链接服务器信息……611
27.1.4 删除链接服务器……611
27.2 链接服务器登录名……612
27.2.1 添加链接服务器登录名映射……612
27.2.2 查看链接登录名……613
27.2.3 删除链接服务器登录名映射……613
27.3 执行分布式查询……614
27.3.1 在链接服务器上执行分布式查询……614
27.3.2 创建和使用引用4部分组成的链接服务器名的别名……615
27.3.3 使用OPENQUERY执行分布式查询……616
27.3.4 使用OPENROWSET执行临时的查询……617

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

27.3.5 使用OPENROWSET BULK选项从文件中读取数据……618
第28章 查询性能调优……621
28.1 查询性能技巧……622
28.2 捕捉和评估查询性能……623
28.2.1 使用sys.dm_exec_requests捕捉执行的查询……623
28.2.2 使用T-SQL命令查看评估的查询执行计划……624
28.2.3 查看执行运行时信息……627
28.2.4 查看性能统计信息和缓存的查询计划……629
28.2.5 基于查询或计划模式查看聚合的性能统计信息……630
28.2.6 标识出最大瓶颈……632
28.2.7 通过数据库和文件标识出I/O争用……633
28.3 索引调优……634
28.3.1 显示索引碎片……635
28.3.2 显示索引使用情况……637
28.4 统计信息……638
28.4.1 手动创建统计信息……639
28.4.2 行子集上创建统计信息……640
28.4.3 更新统计信息……640
28.4.4 生成及更新所有表的统计信息……641
28.4.5 查看详细的统计信息……642
28.4.6 删除统计信息……643
28.5 杂项技术……643
28.5.1 使用动态SQL的替代方法……644
28.5.2 强制SQL Server使用查询计划……646
28.5.3 不修改应用程序的SQL去应用提示……648
28.5.4 从缓存创建计划指南……651
28.5.5 检查计划指南的有效性……653
28.5.6 使用计划指南将非参数化查询参数化……653
28.5.7 限制查询资源消耗竞争……656
第29章 备份与恢复……662
29.1 创建备份和恢复计划……662
29.2 生成备份……663
29.2.1 执行基本的完全备份……665
29.2.2 压缩备份……666
29.2.3 命名和描述备份和媒体……668
29.2.4 配置备份保持期……669
29.2.5 条带化备份集……670
29.2.6 使用命名的备份设备……670
29.2.7 镜像备份集……672
29.2.8 执行事务日志备份……673
29.2.9 不破坏备份序列创建备份……674
29.2.10 执行差异备份……675
29.2.11 备份单个文件或文件组……675
29.2.12 执行部分备份……677
29.2.13 查看备份元数据……678
29.3 还原数据库……679
29.3.1 从完全备份还原数据库……679
29.3.2 从事务日志备份还原数据库……682
29.3.3 从差异备份还原数据库……684
29.3.4 还原文件或文件组……685
29.3.5 执行部分（PARTIAL）还原……686
29.3.6 还原页面……687
29.3.7 用多种恢复路径识别数据库……688
索引……691

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第1章 SELECT



这一章介绍了使用SELECT语句从SQL Server数据库中返回数据的一些技巧。你会注意到，在每一章的开始都会先介绍最基础的概念。这些内容是为那些SQL Server 2008 T-SQL查询语言的初学者准备的。除基础内容之外，还会介绍一些在日常的开发和管理工作中用得到的技巧。这些技巧还会帮助你学习SQL Server 2008中引入的新特性。

书中的大部分示例使用了AdventureWorks数据库（SQL Server 2008 OLTP版），你可以从CodePlex网站（www.codeplex.com）的“Microsoft SQL Server Product Samples: Database”项目中查找名为AdventureWorks2008.msi的文件，然后下载该文件。而且，如果你决定尝试本书技巧的示例，我强烈建议你在非生产性的学习环境下做这些。这样你就可以自由地做试验且不会带来负面影响。

简明扼要是本书的关键原则，因此在最开始描述一个新的T-SQL概念时，我只着重讲解示例代码必要的语法块。在阐明一个概念或任务时，如果语法块对示例来说不是必需的，我就不会讲它。对于完整语法，可以参考联机丛书，因此我不会给你复习全部语法，而只关注技巧中用到的语法。关于技巧返回的结果集，经常会为了节省篇幅而减少显示的列和行。

SQL Server 2008的新特性会贯穿在本书中。对于那些非常重要的改进内容，我会在每章的开头加以声明，这样你就知道去留意它们。本章我会讲到的SQL Server 2008新特性包括：

- GROUP BY子句的新扩展，可以在不需使用UNION ALL的情况下，在同一查询中产生多个分组结果集；
- 声明时初始化变量的新方法，可以减少需要设置变量值的代码。

你几乎可以按任意顺序阅读书中的这些技巧。可以直接跳到感兴趣的主题，也可以按顺序阅读。如果看到有用的内容（代码段或者示例），可以根据自己的应用进行修改或者整合到你自己的函数或存储过程中去，那就体现了本书的价值。

1.1 基本的 SELECT 语句

SELECT命令是T-SQL语言的基础，可以使用它从SQL Server数据库中（更准确地说是从SQL Server数据库中的数据对象中）获取数据。把SELECT语句的完整语法写出来有点儿吓人，但基本的语法可以用如下精炼的形式表示：

```
SELECT select_list
FROM table_list
```

其中select_list参数是你希望在查询的结果中返回的列，table_list参数是提供数据的实际的表或视图。

后面的几个技巧会阐述如何使用基本的SELECT语句。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

1.1.1　从表中选择指定列

这个示例演示了一个对AdventureWorks数据库非常简单的SELECT查询，结果返回3列，显示了HumanResources.Employee表的几行内容。在查询中我们使用了显式列名：

```
USE AdventureWorks
GO

SELECT NationalIDNumber,
       LoginID,
       JobTitle
FROM HumanResources.Employee
```

查询返回的部分结果如下：

```
NationalIDNumber   LoginID                  JobTitle
295847284          adventure-works\ken0     Chief Executive Officer
245797967          adventure-works\terri0   Vice President of Engineering
509647174          adventure-works\roberto0 Engineering Manager
112457891          adventure-works\rob0     Senior Tool Designer
...
954276278          adventure-works\rachel0  Sales Representative
668991357          adventure-works\jae0     Sales Representative
134219713          adventure-works\ranjit0  Sales Representative

(290行受影响)
```

解析

代码的第一行设置了查询所关联的数据库。当第一次登录到SQL Server Management Studio(SSMS)的时候，初始数据库被定义为登录的默认数据库。USE加上数据库名称改变了所操作的数据库：

```
USE AdventureWorks
GO
```

随后是SELECT查询。下面的几行代码定义了哪些列会显示在查询结果中：

```
SELECT NationalIDNumber,
       LoginID,
       JobTitle
```

下一行代码是FROM子句：


```
FROM HumanResources.Employee
```

FROM子句用来指定数据源，在这个示例中是个表。注意表名HumanResources.Employee由两部分组成。第一部分（句点前的部分）为架构（schema），第二部分（句点后的部分）是实际的表名。架构包含了对象，用户拥有的是架构。因为用户拥有架构，并且架构包含对象，所以你不需要修改对象的所有关系就可以改变架构的拥有者。

1.1.2　从所有行中选择所有列

如果希望从FROM子句指定的数据源中显示所有列，可以使用下面的查询：

```
USE AdventureWorks
GO
SELECT   *
FROM   HumanResources.Employee
```

部分的行和列输出显示如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
BusinessEntityID   NationalIDNumber   LoginID                    OrganizationNode
1                  295847284          adventure-works\ken0       0x
2                  245797967          adventure-works\terri0     0x58
3                  509647174          adventure-works\roberto0   0x5AC0
4                  112457891          adventure-works\rob0       0x5AD6
...
```

解析

星号（*）表示从查询的表或视图的每一行中返回所有列。其他细节在前一节中已经阐述。

记住，作为一个好习惯，显式指定想要获取的列要比使用SELECT *更好。如果在应用程序中使用了SELECT *，那么应用程序可能希望从查询中得到相同的列（以相同的顺序）。如果之后要添加新的列到既有的表或视图中，或者对表的列重新排序，那么可能就不能调用这个应用程序了，因为在结果集中新列是不希望出现的。SELECT *在网络上返回的数据可能比真正需要的要多，这会增加结果集的大小和SQL Server实例的数据检索操作，所以它也可能对性能造成负面影响。对于需要每秒数千次事务的应用来说，结果集返回的列数对性能的影响是不可低估的。

1.2 使用基本 WHERE 子句进行有选择的查询

在SELECT查询中，WHERE子句用来限制查询结果集返回的行。简化的包含WHERE子句的语法如下：

```
SELECT select_list
FROM table_list
[WHERE search_conditions]
```

WHERE子句使用了一些搜索条件，它们确定了查询返回的行。搜索条件使用了谓词，它们是计算结果为TRUE、FALSE或UNKNOWN的表达式。

在搜索条件中有NULL数据时可能会出现UNKNOWN值。NULL值不同于空白或零值，只表示值未知。并且，两个NULL值并不相等，不产生UNKNOWN值的话是不能做比较的。

下面的几个技巧将阐述如何使用WHERE子句指定结果集中需要或者不需要返回哪些行。

1.2.1 使用 WHERE 子句指定结果集中返回的行

这个基础的例子示范了如何选择查询结果中返回哪些行：

```
SELECT   Title,
   FirstName,
   LastName
FROM   Person.Person
WHERE Title = 'Ms.'
```

这个示例返回了下面的结果（部分）：

```
Title   FirstName   LastName
Ms.     Gail        Erickson
Ms.     Janice      Galvin
Ms.     Jill        Williams
Ms.     Catherine   Abel
...
Ms.     Abigail     Coleman
Ms.     Angel       Gray
Ms.     Amy         Li

(415行受影响)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个示例中，只有那些Title为Ms.的行被返回。搜索条件在查询的WHERE子句中指定：

```
WHERE Title = 'Ms.'
```

虽然在这个例子中只有一个搜索条件，但在单个查询中几乎可以使用无限多的搜索条件。在下一个技巧中会看到这些内容。

1.2.2　组合搜索条件

本技巧将示范通过使用AND、OR和NOT逻辑运算符来联结多个搜索条件。AND逻辑运算符联结两个以上搜索条件，并且返回每个搜索条件都为真的行。OR逻辑运算符联结两个以上搜索条件，并且返回任一搜索条件为真的行。

在下面第一个例子中，我们在WHERE子句中使用了被AND运算符分隔的两个搜索条件。AND意味着，对于某个行，如果它要在结果集中返回，那么两个搜索条件一定都为真。

```
SELECT   Title,
         FirstName,
         LastName
FROM     Person.Person
WHERE Title = 'Ms.' AND
         LastName = 'Antrim'
```


上面代码返回了如下的结果：

```
Title   FirstName    LastName
Ms.     Ramona       Antrim

(1行受影响)
```

在第二个示例中，我们使用了OR运算符来替代AND，这意味着，只要任一搜索条件为TRUE，该行就会被返回。

```
SELECT   Title,
         FirstName,
         LastName
FROM     Person.Person
WHERE Title = 'Ms.' OR
         LastName = 'Antrim'
```

返回的（部分）结果如下：

```
Title   FirstName   LastName
Ms.     Gail        Erickson
Ms.     Janice      Galvin
...
Ms.     Ramona      Antrim
...
Ms.     Abigail     Coleman
Ms.     Angel       Gray
Ms.     Amy         Li

(415行受影响)
```

解析

在第一个示例中，两个搜索条件使用AND运算符联结：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
WHERE Title = 'Ms.' AND
      LastName = 'Antrim'
```

当为查询增加搜索条件时，可以使用逻辑运算符AND和OR来联结。例如，如果Title等于Ms.并且LastName等于Antrim，那么满足条件的行就会被返回。AND运算符表示，对于要返回的行，两个联结的搜索条件必须都为真。

另外一方面，OR运算符表示，只要任一搜索条件为TRUE就返回行，就像第三个示例演示的那样：

```
WHERE Title = 'Ms.' OR
      LastName = 'Antrim'
```

因此，不像前一个查询只返回了一行，所有Title为Ms.或者LastName为Antrim的行都被返回。

1.2.3 否定搜索条件

NOT逻辑运算符不像AND和OR，它不是用来联结搜索条件的，而是用来否定跟在它后面的表达式的。

下一个示例演示了使用NOT运算符来反转它后面的搜索条件的结果，并限定Title等于Ms.（这就意味着匹配任何除Ms.之外的字符串）：

```
SELECT   Title,
         FirstName,
         LastName
FROM     Person.Person
WHERE NOT Title = 'Ms.'
```

这个示例返回了下面的结果（部分）：

```
Title   FirstName    LastName
Mr.     Josef        Goldberg
Mr.     Hung-Fu      Ting
Mr.     Brian        Welcker
Mr.     Tete         Mensa-Annan
Mr.     Syed         Abbas
Mr.     Gustavo      Achong
Sr.     Humberto     Acevedo
Sra.    Pilar        Ackerman
...
```

解析

这个例子示范了NOT运算符：

```
WHERE NOT Title = 'Ms.'
```

NOT指定搜索条件的反转，在这个例子中指定了仅返回那些Title不等于Ms.的行。

1.2.4 保持 WHERE 子句无歧义

你可以在单个WHERE子句中使用多个运算符（AND、OR、NOT），但重要的是需要适当地把AND和OR放在圆括号中来保证目的明确。AND运算符限制了结果集，OR运算符扩展了返回行的条件。当在同一WHERE子句中使用多个运算符时，运算符优先级会来决定如何计算搜索条件。例如，NOT运算符的优先级（最先运算）比AND高，AND运算符的优先级比OR运算符高。当在同一WHERE子句中同时使用AND和OR运算符时，如果不用括号可能会导致意想不到的结果。

例如，下面的查询可能返回预期外的结果：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SELECT   Title,
   FirstName,
   LastName
FROM   Person.Person
WHERE  Title = 'Ms.' AND
   FirstName = 'Catherine' OR
   LastName = 'Adams'
```

6

这个示例返回了下面的结果（部分）：

```
Title    FirstName    LastName
NULL     Jay          Adams
Ms.      Catherine    Abel
Ms.      Frances      Adams
Ms.      Carla        Adams
Mr.      Jay          Adams
Mr.      Ben          Adams
Ms.      Catherine    Whitney
...
```

对于这段查询，作者的意图是返回Title为Ms.，并且FirstName为Catherine或LastName为Adams的所有行吗？或是作者希望搜索FirstName为Catherine且Title为Ms.，或LastName为Adams的所有人吗？

同时使用了AND和OR的查询应该使用括号来明确哪些行应该被返回。例如，下一查询返回任何Title为Ms.并且FirstName等于Catherine的人。它也将返回其他那些LastName为Adams的人——不管Title和FirstName为何值：

```
SELECT  Title,
        FirstName,
        MiddleName,
        LastName
FROM   Person.Person
WHERE   (Title = 'Ms.' AND
      FirstName = 'Catherine') OR
      LastName = 'Adams'
```

解析

使用圆括号来明确WHERE子句中的多个运算符。圆括号使查询变得更清晰，因为它们可以帮助SQL Server来明确表达式执行的次序。圆括号内的搜索条件是按照从内到外的顺序计算的，所以在本技巧的例子中，下面的搜索条件将先被执行：

```
(Title = 'Ms.' AND
FirstName = 'Catherine')
```

然后执行括号外的OR搜索条件：

```
LastName = 'Adams'
```

1.3 使用运算符和表达式

至此，我们已经接触了=（等于）运算符来指明结果集中的列应该为哪些值。比较运算符 = 检验两个表达式是否相等。*表达式*（expression）是标识符、值和运算符的组合，SQL Server 2005 可以对其求值以获取结果（例如，返回TRUE或FALSE或UNKNOWN）。

7

表1-1列出了一些你可以在搜索条件中使用的运算符。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表1-1 运算符

运算符	描 述
!=	测试某个表达式是否不等于另一个表达式
!>	测试左边条件是否不大于右边表达式
!<	测试左边条件是否不小于右边表达式
<	测试左边条件是否小于右边条件
<=	测试左边条件是否小于或等于右边条件
<>	测试两个表达式是否不相等
=	测试两个表达式是否相等
>	测试左边条件是否大于右边表达式
>=	测试左边条件是否大于或等于右边表达式
ALL	当与比较运算符和子查询一起使用时，假如所有返回值满足搜索条件，相应的行将被返回
ANY	当与比较运算符和子查询一起使用时，假如任意一个返回值满足搜索条件，相应的行将被返回
BETWEEN	指定范围值。在起始值和结束值之间和AND子句一起使用
CONTAINS	为单词或短语执行模糊搜索
ESCAPE	指示通配符应当解释为文本值，而不是通配符的字符。它与LIKE运算符一起使用。例如，在使用LIKE时的模式搜索中，百分号（%）、下划线（_），以及方括号（[]）都有通配符含义。如果要显式地查找实际的百分号，就必须定义ESCAPE字符将其放在通配符之前，指示它为文本值
EXISTS	当在子查询中使用时，EXISTS测试子查询中的行是否存在
FREETEXT	根据意思而非文本值来搜索字符数据中的单词
IN	为搜索条件提供列表值
IS NOT NULL	检查值是否不为空
IS NULL	检查值是否为空
LIKE	检测字符串是否模式匹配
NOT BETWEEN	指定不包含的值的范围。在起始值和结束值之间与AND子句一起使用
NOT IN	提供一个列表值，返回的行不在其中
NOT LIKE	检测字符串是否没有这些模式匹配
SOME	当与比较运算符和子查询一起使用时，如果任何检索值满足搜索条件，相应的行将被返回

正如在表1-1中所见到的，SQL Server 2008包括一些可以在查询表达式中使用的运算符。具体来说，在WHERE子句的上下文中，运算符能用来比较两个表达式，并且检测条件是否为TRUE、FALSE或UNKNOWN。

注解 SQL Server 2008也引入了新的赋值运算符，具体参见第2章。

在下面的几节中将阐述如何在搜索表达式中使用不同的运算符。

1.3.1 在日期范围搜索中使用 BETWEEN

这个例子阐述了使用BETWEEN运算符来指定发生在2002-07-28与2002-07-29之间的销售订单：

```
SELECT   SalesOrderID,
      ShipDate
FROM Sales.SalesOrderHeader
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
WHERE ShipDate BETWEEN '7/28/2002 00:00:00' AND '7/29/2002 23:59:59'
```

查询返回了下面的（部分）结果：

```
SalesOrderID   ShipDate
46845          2002-07-28 00:00:00.000
46846          2002-07-28 00:00:00.000
46847          2002-07-28 00:00:00.000
46848          2002-07-28 00:00:00.000
46849          2002-07-28 00:00:00.000
46850          2002-07-28 00:00:00.000
46851          2002-07-28 00:00:00.000
46852          2002-07-28 00:00:00.000
46853          2002-07-28 00:00:00.000
46854          2002-07-28 00:00:00.000
46855          2002-07-29 00:00:00.000
46856          2002-07-29 00:00:00.000
46857          2002-07-29 00:00:00.000
46858          2002-07-29 00:00:00.000
46859          2002-07-29 00:00:00.000
46860          2002-07-29 00:00:00.000
46861          2002-07-29 00:00:00.000

(17行受影响)
```

解析

在练习中，我们使用了BETWEEN运算符来检测列ShipDate的值是否在两个日期之间：

```
WHERE ShipDate BETWEEN '7/28/2002 00:00:00' AND '7/29/2002 23:59:59'
```

1.3.2 使用比较运算符

下面的例子演示了小于运算符<，该查询示例用它来显示标准价格低于$110.00的产品：

```
SELECT   ProductID,
         Name,
         StandardCost
FROM   Production.Product
WHERE    StandardCost < 110.0000
```


查询返回了如下的（部分）结果：

```
ProductID   Name                      StandardCost
1           Adjustable Race           0.00
2           Bearing Ball              0.00
3           BB Ball Bearing           0.00
4           Headset Ball Bearings     0.00
...
994         LL Bottom Bracket         23.9716
995         ML Bottom Bracket         44.9506
996         HL Bottom Bracket         53.9416

(317行受影响)
```

解析

这个例子演示了使用小于运算符<来返回StandardCost小于110.0000的所有行：

```
WHERE    StandardCost < 110.0000
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

1.3.3 检测 NULL 值

下一个查询检测指定列的NULL值。NULL值不是指值为空或零，而仅代表该值未知。这个查询返回所有产品重量未知的行：

```
SELECT   ProductID,
         Name,
         Weight
FROM     Production.Product
WHERE    Weight IS NULL
```

查询返回了如下的（部分）结果：

```
ProductID   Name                    Weight
1           Adjustable Race         NULL
2           Bearing Ball            NULL
3           BB Ball Bearing         NULL
4           Headset Ball Bearings   NULL
...

(299行受影响)
```

解析

这个例子演示了通过IS NULL运算符来返回所有Weight值未知（不可用）的行：

```
WHERE    Weight IS NULL
```


1.3.4 基于一组值返回行

在这个示例中，IN运算符验证了Color列与一组表达式的相等性：

```
SELECT   ProductID,
      Name,
      Color
FROM Production.Product
WHERE Color IN ('Silver', 'Black', 'Red')
```

查询返回了如下的（部分）结果：

```
ProductID   Name                            Color
317         LL Crankarm                     Black
318         ML Crankarm                     Black
319         HL Crankarm                     Black
...
725         LL Road Frame - Red, 44         Red
739         HL Mountain Frame - Silver, 42  Silver
(174行受影响)
```

解析

这个例子演示了使用IN运算符来返回颜色为Silver、Black或Red的所有产品：

```
WHERE Color IN ('Silver', 'Black', 'Red')
```

1.3.5 LIKE 和通配符结合使用

通配符用在搜索表达式中，查找字符串中匹配的模式。在SQL Server 2008中，有如表1-2所示的通配符选项。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表1-2 通配符

通配符	用 途
%	表示包含零个或多个字符的任意字符串
_	表示任何单个字符
[]	指定某个范围或列表中的一个字符
[^]	指定不在特定范围中的一个字符

下例演示了使用%通配符的LIKE操作，搜索名称从字母B开始的所有产品：

```
SELECT   ProductID,
         Name
FROM   Production.Product
WHERE Name LIKE 'B%'
```

11 上面的代码返回了如下结果：

```
ProductID   Name
3           BB Ball Bearing
2           Bearing Ball
877         Bike Wash - Dissolver
316         Blade

(4行受影响)
```

如果你希望在字符列中搜索百分号%或下划线_的文本值，该怎么做？可以使用ESCAPE运算符（之前在表1-1中描述过）。

下一个查询搜索所有产品名中带有下划线（_）的产品。ESCAPE运算符允许以实际字符搜索通配符。下面将先修改Production.ProductDescription表中的一行，给Description列添加一个百分号：

```
UPDATE Production.ProductDescription
SET Description = 'Chromoly steel. High % of defects'
WHERE ProductDescriptionID = 3
```

然后，查询这张表，查找所有包含百分号文本值的Description：

```
SELECT ProductDescriptionID,Description
FROM Production.ProductDescription
WHERE Description LIKE '%/%%' ESCAPE '/'
```

查询返回：

```
ProductDescriptionID    Description
3                       Chromoly steel. High % of defects
```

解析

通配符允许你搜索基于字符的列中的模式。在本技巧的例子中，%用来表示由零或多个字符构成的字符串：

```
WHERE Name LIKE 'B%'
```

如果要以文本值形式搜索，而不是被SQL Server解释为通配符，可以使用ESCAPE关键字。本技巧的示例是在Description列中搜索文本值百分号：

```
WHERE Description LIKE '%/%%' ESCAPE '/'
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

单引号中的斜线放在ESCAPE命令之后。这表示斜线符号为前面LIKE表达式字符串中的转义字符。如果转义字符出现在搜索条件中的下划线之前，那么它将被视为文本值而不是通配符。

1.3.6 声明变量及为变量赋值

贯穿本书，你会看到在查询中和基于模块的SQL Server对象（存储过程和触发器等）中使用变量的示例。变量是为了临时容纳数据而创建的对象。可以将变量定义成几个不同的数据类型，然后在允许的类型上下文中引用它。

这个技巧将演示使用变量来保存搜索字符串。你会看到两种创建变量并为其赋值的方法。第一个查询演示了SQL Server 2008之前版本的方法：

```
DECLARE @AddressLine1 nvarchar(60)
SET @AddressLine1 = 'Heiderplatz'

SELECT AddressID, AddressLine1
FROM Person.Address
WHERE AddressLine1 LIKE '%' + @AddressLine1 + '%'
```

这个示例中的查询返回地址中包含搜索字符串值的所有行：

```
AddressID   AddressLine1
20333       Heiderplatz 268
17062       Heiderplatz 268
24962       Heiderplatz 662
...
19857       Heiderplatz 948
25583       Heiderplatz 948
28939       Heiderplatz 948
16799       Heiderplatz 978

(18行受影响)
```

在SQL Server 2008中，可以删除SET指令，仅使用DECLARE语句为变量赋值，这样可以减少必需的T-SQL代码：

```
SELECT AddressID, AddressLine1
FROM Person.Address
WHERE AddressLine1 LIKE '%' + @AddressLine1 + '%'
```

从表面上看，这个增强的效果并不明显。但是，如果声明并设置数百个变量，代码的减少量会很明显。

解析

第一个查询用@符号作为前缀声明新变量，随后是定义用来包含搜索字符串的数据类型：

```
DECLARE @AddressLine1 nvarchar(60)
```

在声明变量之后，使用SET命令为该变量赋值（也可以使用SELECT完成这个工作）：

```
SET @AddressLine1 = 'Heiderplatz'
```

然后，在SELECT查询的WHERE子句中使用填充的搜索值，将该变量嵌入在两个%通配符之间，查找地址中包含搜索字符串的所有行：

```
WHERE AddressLine1 LIKE '%' + @AddressLine1 + '%'
```

下一个作用相同的查询则利用了SQL Server 2008的特性，在DECLARE语句中为变量赋值：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DECLARE @AddressLine1 nvarchar(60) = 'Heiderplatz'
```

注解　第2章将展示如何使用SQL Server 2008中新增的赋值运算符来连接赋值操作，它允许行内数据值修改。

1.4　数据分组

GROUP BY子句可以在SELECT查询中使用，用来确定行应该如何分组。简化的语法如下：

```
SELECT select_list
FROM table_list
[WHERE search_conditions]
[GROUP BY group_by_list]
```

GROUP BY跟在可选的WHERE子句后，当我们在SELECT语句中使用聚合函数的时候会经常用到（聚合函数的更多细节请参见第8章）。

1.4.1　使用 GROUP BY 子句

本例使用GROUP BY子句统计Sales.SalesOrderHeader表的某段日期中每天的付款总数：

```
SELECT   OrderDate,
   SUM(TotalDue) TotalDueByOrderDate
FROM   Sales.SalesOrderHeader
WHERE OrderDate BETWEEN '7/1/2001' AND '7/31/2001'
GROUP BY OrderDate
```

这个查询返回了如下（部分）结果：

```
OrderDate                TotalDueByOrderDate
2001-07-01 00:00:00.000  665262.9599
2001-07-02 00:00:00.000  15394.3298
2001-07-03 00:00:00.000  16588.4572
...
2001-07-30 00:00:00.000  15914.584
2001-07-31 00:00:00.000  16588.4572

(31行受影响)
```

解析

在此技巧的例子中，我们在SELECT查询中使用了GROUP BY子句来确定行应该如何分组。先来看查询的第一行，SELECT子句指定了需要返回的OrderDate以及TotalDue列值的总和。SUM为聚合函数。聚合函数对一组值（在这个例子中为TotalDue）进行计算，返回一个值（基于某个OrderDate的TotalDue的总和）：


```
SELECT   OrderDate,
   SUM(TotalDue) TotalDueByOrderDate
```

注意，在SUM(TotalDue)聚合过程中使用了列别名（column alias）。列别名为一个计算列、聚合列或普通列返回不同的名称。在查询中的下一部分，表Sales.SalesOrderHeader在FROM子句中被引用：

```
FROM   Sales.SalesOrderHeader
```

下一步，通过限定OrderDate来返回2001年7月的行：

```
WHERE OrderDate BETWEEN '7/1/2001' AND '7/31/2001'
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

结果集通过OrderDate进行分组（注意，分组可以发生在一列或多列上）：

```
GROUP BY OrderDate
```

如果从查询中删除GROUP BY子句，那么在SELECT子句中使用聚合函数将会引发下面的错误：

```
消息8120，级别16，状态1，第1行
选择列表中的列'Sales.SalesOrderHeader.OrderDate'无效，
因为该列没有包含在聚合函数或GROUP BY子句中，
```

引发这个错误的原因是没有在SELECT列表中的聚合函数内使用的列必须在GROUP BY子句中列出。

1.4.2 使用 GROUP BY ALL

在GROUP BY后增加ALL关键字，分组就会使用所有行的值，即使WHERE子句限定了这些行不能显示。

除了ALL子句，这个示例执行的查询几乎与前一个示例一样：

```
SELECT   OrderDate,
         SUM(TotalDue) TotalDueByOrderDate
FROM   Sales.SalesOrderHeader
WHERE OrderDate BETWEEN '7/1/2001' AND '7/31/2001'
GROUP BY ALL OrderDate
```

这个查询返回的（部分）结果如下：

```
OrderDate                TotalDueByOrderDate
2002-08-12 00:00:00.000  NULL
2003-07-25 00:00:00.000  NULL
2004-06-21 00:00:00.000  NULL
2001-07-22 00:00:00.000  42256.626
警告：聚合或其他SET操作消除了空值，

(1124行受影响)
```


解析

在GROUP BY ALL示例返回的结果中，那些没有包含在WHERE子句限定的日期内记录的TotalDueByOrderDate为NULL。这并不意味着它们没有任何行，而是数据没有返回。

这个查询也返回一个警告，如下所示：

```
警告：聚合或其他SET操作消除了空值，
```

它的意思是SUM聚合遇到了NULL值，而且没把它们包含在总和中。对于SUM聚合函数来说，一切正常；然而NULL值在其他聚合函数中会导致意想不到的结果。例如：AVG函数忽略NULL值，但COUNT函数不忽略。如果你的查询同时使用了这两个函数，你可能会认为包含在COUNT中的NULL值影响了AVG的结果——但其实没有。

1.4.3 使用 HAVING 选择性地查询分组的数据

SELECT语句中的HAVING子句允许你在使用了GROUP BY和（或）聚合值的查询中指定一个搜索条件。语法如下：

```
SELECT select_list
FROM table_list
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
[ WHERE search_conditions ]
[ GROUP BY group_by_list ]
[ HAVING search_conditions ]
```

HAVING子句在GROUP BY子句之后使用，用来进一步限定结果。而WHERE子句用来限定数据聚合或分组之前返回的行。与WHERE相比，HAVING限定数据被分组或聚合之后聚合的数据。

这个示例查询了两张表：Production.ScrapReason和Production.WorkOrder。表Production.ScrapReason是包含制造失败原因的查找表，而表Production.WorkOrder包含了生产制作顺序，它控制哪些产品需要按时按量进行生产来满足库存和销售需求。

这个示例向管理层报告了哪些“失败原因”发生了50次或更多次：

```
SELECT  s.Name,
   COUNT(w.WorkOrderID) Cnt
FROM Production.ScrapReason s
INNER JOIN Production.WorkOrder w ON
   s.ScrapReasonID = w.ScrapReasonID
GROUP BY s.Name
HAVING COUNT(*)>50
```

这个查询返回：

```
Name                              Cnt
Gouge in metal                    54
Stress test failed                52
Thermoform temperature too low    63
Trim length too long              52
Wheel misaligned                  51
(5行受影响)
```

16

解析

在这个技巧中，SELECT子句请求各种失败原因的WorkOrderID计数：

```
SELECT  s.Name,
      COUNT(w.WorkOrderID)
```

通过ScrapReasonID列联结两张表：

```
FROM Production.ScrapReason s
INNER JOIN Production.WorkOrder w ON
   s.ScrapReasonID = w.ScrapReasonID
```

由于在SELECT子句中使用了聚合函数，未聚合的列必须出现在GROUP BY子句中：

```
GROUP BY s.Name
```

最后，使用HAVING查询确定，对于被选择和分组的数据，结果集中只有数量为50或更多的行才会被返回。

```
HAVING COUNT(*)>50
```

1.5 对结果排序

ORDER BY子句根据指定的列或表达式来对查询结果进行排序。ORDER BY的基本语法如下：

```
SELECT select_list
FROM table_list
[WHERE search_conditions]
[GROUP BY group_by_list]
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
[HAVING search_conditions]
[ORDER BY order_list [ASC | DESC] ]
```

ORDER BY必须在FROM子句之后使用，可选的WHERE、GROUP BY和HAVING子句也是如此。

1.5.1 使用 ORDER BY 子句

这个例子演示了根据ProductID和EndDate列对查询结果进行排序：

```
SELECT   p.Name,
      h.EndDate,
      h.ListPrice
FROM   Production.Product p
INNER JOIN Production.ProductListPriceHistory h ON
   p.ProductID = h.ProductID
ORDER BY p.Name, h.EndDate
```

这个查询返回：

```
Name                    EndDate                 ListPrice
All-Purpose Bike Stand  NULL                    159.00
AWC Logo Cap            NULL                    8.99
AWC Logo Cap            2002-06-30 00:00:00.000 8.6442
AWC Logo Cap            2003-06-30 00:00:00.000 8.6442
Bike Wash - Dissolver   NULL                    7.95
Cable Lock              2003-06-30 00:00:00.000 25.00
Chain                   NULL                    20.24
...
(395行受影响)
```


ORDER BY默认的排序方式是升序，可以使用ASC来显式声明。在下一个示例中，DESC用来以相反的次序（降序）返回结果：

```
SELECT   p.Name,
      h.EndDate,
      h.ListPrice
FROM   Production.Product p
INNER JOIN Production.ProductListPriceHistory h ON
   p.ProductID = h.ProductID
ORDER BY p.Name DESC, h.EndDate DESC
```

这个查询返回了如下部分结果：

```
Name                    EndDate                 ListPrice
Women's Tights, S       2003-06-30 00:00:00.000 74.99
Women's Tights, M       2003-06-30 00:00:00.000 74.99
...
AWC Logo Cap            2002-06-30 00:00:00.000 8.6442
AWC Logo Cap            NULL                    8.99
All-Purpose Bike Stand  NULL                    159.00

(395行受影响)
```

第三个例子演示了基于一个没有在SELECT子句中使用的列进行排序的结果：

```
SELECT   p.Name
FROM   Production.Product p
ORDER BY p.Color
```

查询返回的部分结果如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
name
Guide Pulley
LL Grip Tape
ML Grip Tape
HL Grip Tape
Thin-Jam Hex Nut 9
...
```

解析

尽管不使用ORDER BY子句查询有时似乎也可以返回正确的数据，聚集索引（在第5章查看聚集索引更详细的信息）中的物理键列顺序决定了自然排序的结果。如果结果集中行的顺序是重要的，你一定不要依赖隐式的物理顺序。如果结果集需要排序，则一定要使用ORDER BY。

第一个例子查询了Production.Product和Production.ProductListPriceHistory表，以按时间顺序查看产品价格的历史数据。

注解 1.8.1节会提供完整的INNER JOIN信息。

下面的代码行先按产品名（字母顺序）排列结果，后按结束日期排列结果：

```
ORDER BY p.Name, h.EndDate
```

可以在ORDER BY子句中指定一个或多个列，只要所有列总共不超过8 060字节。

第二个例子示范了以降序的顺序（默认的顺序是升序）返回结果。在每一个需要降序排序的列后面引用DESC关键字：

```
ORDER BY p.Name DESC, h.EndDate DESC
```

第三个例子演示了通过SELECT语句中未使用的列对结果排序：

```
ORDER BY p.Color
```

需要提醒的是，如果根据没有选择的列进行排序并且指定了SELECT DISTINCT，那么ORDER BY的列必须出现在选择列表中。

1.5.2 在排序的结果中使用 TOP 关键字

TOP关键字允许根据定义的行的数量或百分比查询出开始的*n*行。查询如何排序也将影响返回的这些行。

注解 SQL Server也提供了可用来按结果集的分区排列每一行的排名函数。排名函数的介绍请看第8章。

在这个例子中，从Purchasing.Vendor表中返回前10条CreditRating列为最大值的行：

```
SELECT TOP 10 v.Name,
   v.CreditRating
FROM Purchasing.Vendor v
ORDER BY v.CreditRating DESC, v.Name
```

这个查询返回：

```
Name                    CreditRating
Merit Bikes             5
Victory Bikes           5
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Proseware, Inc.              4
Recreation Place             4
Consumer Cycles              3
Continental Pro Cycles       3
Federal Sport                3
Inner City Bikes             3
Northern Bike Travel         3
Trey Research                3

(10行受影响)
```

下一个例子演示了在查询中使用局部变量限制返回行的百分比：

```
DECLARE @Percentage float

SET @Percentage = 1

SELECT TOP (@Percentage) PERCENT
      Name
FROM Production.Product
ORDER BY Name
```

这个例子从Production.Product表中返回前1%以产品名排序的行：

```
Name
Adjustable Race
All-Purpose Bike Stand
AWC Logo Cap
BB Ball Bearing
Bearing Ball
Bike Wash - Dissolver
(6行受影响)
```

解析

在SQL Server的早期版本中，开发者使用SET ROWCOUNT来限制返回或影响的行数。在SQL Server 2005和2008中，TOP通常执行得会更快，所以应该使用TOP关键字取代SET ROWCOUNT。然而，在SQL Server的早期版本中，我们不能在TOP子句中使用局部变量，这也是人们喜欢使用SET ROWCOUNT而不喜欢TOP的一个很重要的原因。现在这些功能上的问题已经不存在了，还有什么理由不使用TOP呢？

提示 TOP关键字现在也可用在INSERT、UPDATE和DELETE语句中，在将来的SQL Server版本中将不再支持SET ROWCOUNT特性。关于TOP和数据修改配合的更多内容，参见第2章。

第一个例子的关键是TOP关键字，TOP后面跟随的是要返回的行数：

```
SELECT TOP 10 v.Name
```

同样重要的是ORDER BY子句，它用来在返回TOP *n*行之前对结果进行排序：


```
ORDER BY v.CreditRating DESC, v.Name
```

第二个例子演示如何使用新的局部变量赋值功能来实现TOP PERCENT：

```
DECLARE @Percentage float

SET @Percentage = 1
SELECT TOP (@Percentage) PERCENT
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

新局部变量的功能允许创建脚本、函数或存储过程，根据调用方设置的值而不是在TOP中写死的一个值来确定查询返回的行。

1.6　SELECT子句技术

SELECT子句主要用来定义返回结果集中的哪些列，但是它的功能不仅限于此。在接下来的几个查询中将详细讲解SELECT子句技术：

- 使用DISTINCT关键字消除重复值；
- 使用列别名重命名列；
- 拼接字符串值到单个列；
- 创建一个能自己创建可执行的T-SQL脚本的SELECT语句；
- 创建以逗号分隔值的一组值。

1.6.1　使用DISTINCT消除重复值

SELECT语句的默认行为是使用ALL关键字（但是因为它是默认的，你很少见到在查询中使用它），这意味着如果有行存在，它们将全部被获取并显示。使用DISTINCT关键字取代ALL允许你只返回结果中（对选定列）不同的行。

这个示例教你如何使用DISTINCT关键字从一组被选择的列中消除重复值，这样只有唯一的行才会出现：

```
SELECT   DISTINCT HireDate
FROM   HumanResources.Employee
```

结果显示了从表HumanResources.Employee返回的所有唯一的雇佣时间：

```
HireDate
1996-07-31 00:00:00.000
1997-02-26 00:00:00.000
1997-12-12 00:00:00.000
1998-01-05 00:00:00.000
...
2002-11-01 00:00:00.000
2003-04-15 00:00:00.000
2003-07-01 00:00:00.000

(164行受影响)
```


解析

使用DISTINCT关键字来返回结果集中不同的值。在这个技巧中，DISTINCT用来返回唯一的HireDate列值。

警告　确认只有在真正需要或必要时使用DISTINCT，因为它会使具有较大结果集的查询速度减慢。

1.6.2　在聚合函数中使用DISTINCT

对于那些用在聚合函数中的列也可以使用DISTINCT（聚合函数的更多细节会在第8章中讲解）。如果想对列的唯一值使用聚合，可以这么做。

例如，如果你希望计算产品列表的平均价格，可以使用下面的查询：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SELECT   AVG(ListPrice)
FROM Production.Product
```

这个查询返回：

```
438.6662
```

但是前面的查询计算的是基于所有产品的平均价格。如果一些产品类型比其他产品类型多很多会怎么样呢？如果你只对唯一零售价的平均值感兴趣呢？

那么，你应该写下面的查询：

```
SELECT   AVG(DISTINCT ListPrice)
FROM Production.Product
```

它先返回唯一的一组零售价，然后计算它们的平均值：

```
437.4042
```

解析

DISTINCT可用来从结果集中返回唯一行，也可强行在聚合函数中指定唯一列值。在这个示例中，DISTINCT关键字被放入聚合函数的括号中。

1.6.3 使用列别名

对于列的计算或聚合函数之类的工作，可以使用列别名显式地命名查询输出的列。也可以使用列别名重命名已经存在名称的列，它可以帮助我们隔离调用的应用程序和基础列（允许在不改变返回的列名的情况下换出基础列）。可以使用关键字AS指定列别名，或是简单地在列或表达式后直接写出列别名。

这个例子演示了使用两种不同的方法产生列别名。

```
SELECT   Color AS 'Grouped Color',
      AVG(DISTINCT ListPrice) AS 'Average Distinct List Price',
      AVG(ListPrice) 'Average List Price'
FROM Production.Product
GROUP BY Color
```

这个查询返回了下面的（部分）结果：

```
Grouped Color   Average Distinct List Price   Average List Price
NULL            65.9275                      16.8641
Black           527.5882                     725.121
Blue            825.2985                     923.6792
Grey            125.00                       125.00
Multi           49.6566                      59.865
Red             1332.6425                    1401.95
Silver          726.2907                     850.3053
Silver/Black    61.19                        64.0185
White           9.245                        9.245
Yellow          991.7562                     959.0913

(10行受影响)
```

解析

这个例子展示了3种使用列别名的情况。第一种情况演示了如何使用AS子句重命名存在的列。AS

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

子句用来改变结果中的列名，或给衍生（计算或聚合之后的）列增加一个名称：

```
SELECT Color AS 'Grouped Color',
```

第二种情况演示了如何为聚合函数增加列名：

```
AVG(DISTINCT ListPrice) AS 'Average Distinct List Price',
```

第三种情况演示了如何不使用AS关键字（它可以被直接省略掉）增加列别名：

```
AVG(ListPrice) 'Average List Price'
```

1.6.4 使用 SELECT 创建脚本

作为一个DBA或开发者，你有时会需要T-SQL脚本在一个数据库中的多个对象或者是跨SQL Server实例的多个数据库上运行。例如，你可能想显示数据库中每个用户表有多少行。或者可能你有一个包含很多列的非常大的表，它们都需要在搜索条件中验证，但是你不希望手动键入每一列。

下一个技巧提供了一种节省时间的技术，即使用SELECT来为我们输出T-SQL。可以修改代码以适应各种目的。

在这个示例中，假设你希望检查表中所有值都是NULL的行。表中有很多列，你不希望手动输入它们。可以创建一段脚本来完成这个工作：

23

```
SELECT column_name + ' IS NULL AND '
FROM INFORMATION_SCHEMA.columns
WHERE table_name = 'Employee'
ORDER BY ORDINAL_POSITION
```

这个查询返回可以整合到WHERE子句中的代码（只需要去除这个WHERE条件尾部的AND）：

```
EmployeeID IS NULL AND
NationalIDNumber IS NULL AND
ContactID IS NULL AND
LoginID IS NULL AND
ManagerID IS NULL AND
Title IS NULL AND
BirthDate IS NULL AND
MaritalStatus IS NULL AND
Gender IS NULL AND
HireDate IS NULL AND
SalariedFlag IS NULL AND
VacationHours IS NULL AND
SickLeaveHours IS NULL AND
CurrentFlag IS NULL AND
rowguid IS NULL AND
ModifiedDate IS NULL AND

(16行受影响)
```

解析

本例使用字符串拼接以及INFORMATION_SCHEMA.columns系统视图来生成表Employee中的很多列。对于每列，IS NULL AND都被拼接到列名之后。结果可以复制到查询的WHERE子句中，允许你查询所有列都是NULL值的行。

这种拼接SQL命令到各种系统数据列的常用技术还有很多种应用方式，包括用于创建对应表或其他数据库对象的脚本。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

警告 当编写对应多个对象或多个数据库的动作脚本时要非常小心——务必确认改变是你所希望的，并且要充分了解脚本的输出。

1.6.5 字符串拼接

字符串拼接操作通过使用+运算符连接两个表达式来实现，例如：

```
SELECT   'The ' +
      p.name +
      ' is only ' +
      CONVERT(varchar(25),p.ListPrice) +
      '!'
FROM   Production.Product p
WHERE   p.ListPrice between 100 AND 120
ORDER BY p.ListPrice
```


这个查询返回：

```
The ML Bottom Bracket is only 101.24!
The ML Headset is only 102.29!
The Rear Brakes is only 106.50!
The Front Brakes is only 106.50!
The LL Road Rear Wheel is only 112.57!
The Hitch Rack - 4-Bike is only 120.00!
```

解析

当使用字符数据类型时，使用+运算符就可以将表达式拼接到一起。在这个示例中，文本值被拼接到表Production.Product的列中。每一行都构成了为低价产品而欢呼的一句话。也可以拼接日期，只要使用CAST或CONVERT将日期转换为字符或可变的字符数据类型即可。

字符串拼接经常用在产生终端用户报表时（比如说在一个列显示姓和名），或者需要将多列数据组合到一列（将在下一节看到）时。

1.6.6 使用 SELECT 创建逗号分隔的列表

下一个示例将会演示如何通过SELECT查询创建逗号分隔的列表。可以通过多种方式使用这个方法。例如，可以把它整合到一个用户定义的函数中，该函数返回一列以逗号分隔的列表来表示销售人员的销售范围（参见第11章）。

这个示例演示了返回一对多表的数据到一个输出字符串中：

```
DECLARE @Shifts varchar(20) = ''

SELECT   @Shifts = @Shifts + s.Name + ','
FROM   HumanResources.Shift s
ORDER BY s.EndTime

SELECT @Shifts
```

这个查询返回：

```
Night,Day,Evening,

(1行受影响)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

脚本的第一部分创建了一个用来保存逗号分隔的列表的局部变量。因为我们不能将字符串与NULL值拼接，所以变量应该赋为初始的空白值，正如这个示例所做的：

25
```
DECLARE @Shifts varchar(20) = ''
```

在查询中，从表HumanResources.Shift中收集按照EndTime排序的轮班人员列表。这个示例最关键的地方是，局部变量被赋值为它本身拼接到轮班人员名，然后拼接一个逗号。这个查询按照EndTime排序遍历每一个值，并且把每一项追加到局部变量后面：

```
SELECT   @Shifts = @Shifts + s.Name + ','
FROM   HumanResources.Shift s
ORDER BY s.EndTime
```

使用SELECT来显示局部变量的最终内容：

```
SELECT @Shifts
```

1.6.7 使用 INTO 子句

使用SELECT语句中的INTO子句，可以基于查询结果的列和行创建一个新表。理想情况下应该使用CREATE TABLE命令创建表。然而，使用INTO提供了一种创建新表的快速方法，不需要显式定义列名和数据类型。

INTO子句允许基于查询返回的列与行在SELECT语句中创建表，INTO语法如下：

```
SELECT select_list
[INTO new_table_name]
FROM table_list
```

INTO子句出现在SELECT子句之后，FROM子句之前，下一个技巧将演示它。

在这第一个例子中，基于查询的结果创建了一个新表：

```
SELECT   BusinessEntityID,
         Name,
         SalesPersonID,
         Demographics
INTO   Store_Archive
FROM   Sales.Store
```

查询返回了插入到新表Store_Archive中的行数，而不是返回查询结果：

```
(701行受影响)
```

在第二个示例中，创建了表但是没有插入任何行：

```
SELECT   BusinessEntityID,
         Name,
         SalesPersonID,
         Demographics
INTO   Store_Archive_2
FROM   Sales.Store
WHERE   1=0
```

查询返回了插入到新表Store_Archive_2中的行数（在这里为0）：

26
```
(0行受影响)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

此技巧的示例看起来像常规的SELECT查询，只是在SELECT和FROM子句之间插入了下面的指令：

```
INTO Store_Archive
```

INTO子句后面是一个表名（之前它必须不存在）。它可以为永久表、临时表或全局临时表（关于这些对象类型的更多详细信息参见第4章）。你选择的列确定了表的结构。

这是一个快速“复制”已存在的表的结构和数据的重要技术。使用INTO，就不需要事先显式定义新表的结构（例如，不需要使用CREATE TABLE语句）。

警告 尽管复制了选择的列的结构，但从属于源表的约束、索引和其他独立的对象没有被复制。

第二个例子创建了一个没有填充行的新表。这是由于使用的WHERE子句条件总是运算为FALSE：

```
WHERE 1=0
```

由于数字1永远不会等于数字0，没有哪一行会运算为TRUE，因此任何一行都不会插入到新表中。尽管如此，新表依然会被创建。

1.7 子查询

子查询是嵌套在另外一个SELECT、INSERT、UPDATE或DELETE语句中的SELECT查询。一个子查询也可以嵌套在另外一个子查询中。子查询经常被重写成普通的JOIN，但是有时检测是否存在值的子查询（在这个技巧中会演示）能比等价的非子查询的方式执行得更好。

相关子查询是指结果依赖于外部查询值的子查询[①]。

使用子查询检验匹配的存在

第一个示例演示了用相关子查询检验匹配行的存在：

```
SELECT DISTINCT s.PurchaseOrderNumber
FROM Sales.SalesOrderHeader s
WHERE EXISTS (    SELECT SalesOrderID
         FROM Sales.SalesOrderDetail
         WHERE UnitPrice BETWEEN 1000 AND 2000 AND
               SalesOrderID = s.SalesOrderID)
```

查询返回了如下（部分）结果：

```
PurchaseOrderNumber
PO8410140860
PO12325137381
PO1160166903
PO1073122178
...
PO15486173227
PO14268145224

(1989行受影响)
```


① 相关子查询是主查询遍历每条查询的记录时都要针对该记录的值执行的子查询，效率比较低。——译者注

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第二个示例演示了普通的非相关子查询：

```
SELECT   BusinessEntityID,
         SalesQuota CurrentSalesQuota
FROM      Sales.SalesPerson
WHERE      SalesQuota =
            (SELECT MAX(SalesQuota)
             FROM Sales.SalesPerson)
```

查询返回了3个具有最大销售额300 000的销售人员：

```
BusinessEntityID   CurrentSalesQuota
275                300000.00
279                300000.00
287                300000.00
警告: 聚合或其他SET操作消除了空值.

(3行受影响)
```

解析

第一个例子的关键部分是WHERE子句中的子查询，它检验UnitPrice在1000和2000之间的产品的SalesOrderID是否存在。在子查询的WHERE子句中，通过声明SalesOrderID = s.SalesOrderID，在外部查询和内部查询之间建立了联结。子查询使用了外部查询返回的每一行中的SalesOrderID。

在第二个例子中，在子查询中没有使用WHERE子句来关联外部表，所以它不是相关子查询。查询返回的值用来参与WHERE子句中的=运算。

1.8 从多个数据源查询

JOIN关键字能让我们将多个表和/或视图中的数据组合到一个结果集中。它将一列或多列从一个表联结到另外一个表，计算它们是否匹配。

使用JOIN关键字，可以根据联结条件来联结两个表。大多数情况下，你会看到有一个联结条件来测试一个表中的一列和第二个表中的另外一列相比是否相等（相联结的列不需要有相同的名字，只需要有可兼容的数据类型）。

提示 作为查询性能的最佳实践，尽量避免在JOIN子句中转换列的数据类型（例如使用CONVERT或CAST）。最好是修改底层的架构来匹配数据类型（或者预先在单独的表、临时表、表变量或者公共表表达式（CTE）中进行数据转换）。在频繁执行的查询中进行隐式数据类型转换会引发明显的性能问题（例如，将nchar转换为char）。

28

SQL Server 2005的联结类型有3种：内联结、外联结和交叉联结。内联结使用INNER JOIN关键字。INNER JOIN通过匹配两个表中共同的值进行操作。只有满足联结条件的行才会用来构建结果集。INNER JOIN是默认的JOIN类型，因此如果你喜欢，可以在INNER JOIN操作中仅使用JOIN关键字。

外联结有3种联结类型：LEFT OUTER、RIGHT OUTER和FULL OUTER联结。LEFT OUTER和RIGHT OUTER联结与INNER JOIN相似，都是返回匹配联结条件的行。不过和INNER JOIN不同的是，LEFT OUTER JOIN返回联结对中第一个表不匹配的行，而RIGHT OUTER JOIN返回联结对中第二个表不匹配的行。FULL OUTER JOIN子句返回左表和右表都不匹配的行。

一个不常使用的联结类型是交叉联结CROSS JOIN。当没有使用WHERE子句的时候，一个CROSS JOIN返回一个笛卡尔积（cartersian product）。笛卡尔积根据左表中的行乘以右表中的行的每一种可能的组

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

合产生结果集。例如，如果Stores表有7行，Sales表有22行，那么在查询结果中你会获取154行（7乘22），显示出行的每一种可能的组合。

在下面几个技巧中我们会演示各种联结类型。

1.8.1 使用内联结

下面这个内联结联结了3个表，目的是返回某个产品的折扣信息：

```
SELECT   p.Name,
         s.DiscountPct
FROM Sales.SpecialOffer s
INNER JOIN Sales.SpecialOfferProduct o ON
         s.SpecialOfferID = o.SpecialOfferID
INNER JOIN Production.Product p ON
         o.ProductID = p.ProductID
WHERE p.Name = 'All-Purpose Bike Stand'
```

这个查询的结果如下：

```
Name                    DiscountPct
All-Purpose Bike Stand  0.00

(1行受影响)
```

解析

联结是在FROM子句的第一个表之后开始。在本示例中，把3个表联结在了一起：Sales.SpecialOffer、Sales.SpecialOfferProduct和Production.Product。FROM子句中引用的第一个表Sales.SpecialOffer包含了一个销售折扣查询：

```
FROM Sales.SpecialOffer s
```

注意表名后的字母s。这是*表别名*（table alias）。一旦在查询中使用了多个表，知道各列属于哪个数据源非常重要。如果在两个不同的表中存在同样的列名，SQL编译器会给出一个错误，要求你明确哪个列是你希望返回的。

作为最佳实践，只要列名在查询中指定，最好使用别名。对于每一个引用的表，每个表名都使用字符来代替——让你不用为每次都拼写表名而烦恼。这个查询使用了一个字符作为表别名，但你可以使用任何有效的标识符。*表别名*不仅能让你缩短或明确原始表名，而且如果你希望使用其他的表或视图名替代它或者需要自联结表，你可以换掉基表名。表别名是可选的，但是如果查询超过1个表则推荐使用别名。表别名跟在FROM子句中表名的后面。由于表别名是可选的，因此也可以在每次引用表中的列时指定整个表名。

再来看示例……INNER JOIN关键字后面紧跟的是第一个表引用，然后是要联结的表的别名：

```
INNER JOIN Sales.SpecialOfferProduct o
```

在这之后，ON关键字用于联结列：

```
ON
```

INNER JOIN是基于两个列的相等性，一列来自第一个表，另外一列来自第二个表：

```
s.SpecialOfferID = o.SpecialOfferID
```

然后，再内联结Production.Product表：

```
INNER JOIN Production.Product p ON
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
   o.ProductID = p.ProductID
```

最后，使用WHERE子句来过滤最终的结果集中返回的行：

```
WHERE Name = 'All-Purpose Bike Stand'
```

1.8.2 使用外联结

这个技巧会比较INNER JOIN和LEFT OUTER JOIN的结果。第一个查询使用Person.StateProvince表和Sales.SalesTaxRate表显示税率状态和省。下面的查询使用INNER JOIN：

```
SELECT   s.CountryRegionCode,
      s.StateProvinceCode,
      t.TaxType,
      t.TaxRate
FROM Person.StateProvince s
INNER JOIN Sales.SalesTaxRate t ON
   s.StateProvinceID = t.StateProvinceID
```

这个查询返回下面的（部分）结果：

```
CountryRegionCode    StateProvinceCode    TaxType    TaxRate
CA                   AB                   1          14.00
CA                   ON                   1          14.25
CA                   QC                   1          14.25
...
FR                   FR                   3          19.60
GB                   ENG                  3          17.50

(29行受影响)
```


但是使用INNER JOIN，只能看到那些在Sales.SalesTaxRate表中有行的Person.StateProvince表中的数据。要查看Person.StateProvince表中的所有行，而不去管它是否有关联的税率，可以使用LEFT OUTER JOIN：

```
SELECT   s.CountryRegionCode,
      s.StateProvinceCode,
      t.TaxType,
      t.TaxRate
FROM Person.StateProvince s
LEFT OUTER JOIN Sales.SalesTaxRate t ON
   s.StateProvinceID = t.StateProvinceID
```

这个查询返回下面的（部分）结果：

```
CountryRegionCode    StateProvinceCode    TaxType    TaxRate
CA                   AB                   1          14.00
CA                   AB                   2          7.00
US                   AK                   NULL       NULL
US                   AL                   NULL       NULL
US                   AR                   NULL       NULL
AS                   AS                   NULL       NULL
US                   AZ                   1          7.75
...
FR                   94                   NULL       NULL
FR                   95                   NULL       NULL
(184行受影响)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

这个技巧的示例演示了INNER JOIN查询和LEFT OUTER JOIN查询的比较。LEFT OUTEER JOIN返回了联结对的第一个表中未匹配的行。注意这个查询是如何为Person.StateProvince表中那些在Sales.SalesTaxRate表中找不到关联行的行返回NULL值的。

1.8.3 使用交叉联结

在这个示例中，我们对Person.StateProvince表和Sales.SalesTaxRate表进行交叉联结：

```
SELECT   s.CountryRegionCode,
         s.StateProvinceCode,
         t.TaxType,
         t.TaxRate
FROM Person.StateProvince s
CROSS JOIN Sales.SalesTaxRate t
```

这个查询返回下面的（部分）结果：

```
CountryRegionCode    StateProvinceCode    TaxType    TaxRate
CA                   AB                   1          14.00
US                   AK                   1          14.00
US                   AL                   1          14.00
US                   AR                   1          14.00
AS                   AS                   1          14.00
...
FR                   94                   3          17.50
FR                   95                   3          17.50

(5249行受影响)
```


解析

没有WHERE子句的CROSS JOIN返回笛卡尔积。CROSS JOIN的结果显示了逻辑上没有关系的StateProvince和SalesTaxRate的信息。由于Person.StateProvince表有181条记录，Sales.SalesTaxRate表有29条记录，因此查询返回5 249条记录。

1.8.4 在同一查询中多次引用同一个表

有时可能会需要将同一个表作为两个独立的表来使用。这可能是因为表中包含嵌套层次的数据（例如，表中包含将管理员ID作为外键的职员记录，而这个管理员ID是引用到职员ID的），或者可能你希望在不同时间段引用同一个表（比较2007年和2008年的销售记录）。

可以通过使用表别名来完成表的自联结。

在这个示例中，在FROM子句中引用了两次Sales.SalesPersonQuotaHistory表，一个引用了2004年的销售定额数据，另一个引用了2003年的销售定额数据：

```
SELECT s.BusinessEntityID,
           SUM(s2004.SalesQuota) Total_2004_SQ,
       SUM(s2003.SalesQuota) Total_2003_SQ
FROM Sales.SalesPerson s
LEFT OUTER JOIN Sales.SalesPersonQuotaHistory s2004 ON
     s.BusinessEntityID = s2004.BusinessEntityID AND
     YEAR(s2004.QuotaDate)= 2004
LEFT OUTER JOIN Sales.SalesPersonQuotaHistory s2003 ON
     s.BusinessEntityID = s2003.BusinessEntityID AND
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    YEAR(s2003.QuotaDate)= 2003
GROUP BY s.BusinessEntityID
```

查询返回如下（部分）结果：

```
BusinessEntityID   Total_2004_SQ    Total_2003_SQ
274                1084000.00       1088000.00
275                6872000.00       9432000.00
276                8072000.00       9364000.00
...
289                8848000.00       10284000.00
290                6460000.00       5880000.00

(17行受影响)
```

解析

这个技巧查询了2004年和2003年的销售定额结果。FROM子句包含了到所有销售人员标识符的“锚”：

```
...
FROM Sales.SalesPerson s
```

然后左外部联结第一个对销售定额数据的引用，它的别名为s2004：

```
LEFT OUTER JOIN Sales.SalesPersonQuotaHistory s2004 ON
    s.BusinessEntityID = s2004.BusinessEntityID AND
    YEAR(s2004.QuotaDate)= 2004
```

32

下一步，创建另一个对同一销售定额表的引用，而这一次将表命名为s2003：

```
LEFT OUTER JOIN Sales.SalesPersonQuotaHistory s2003 ON
    s.BusinessEntityID = s2003.BusinessEntityID AND
    YEAR(s2003.QuotaDate)= 2003
GROUP BY s.BusinessEntityID
```

正如在这里演示的，只要将表用唯一的表别名与另一引用的对象区分开来，你就可以在同一查询中多次引用同一个表。

1.8.5 使用衍生表

衍生表是在FROM子句中作为表的SELECT语句。衍生表有些时候可以提供比临时表更好的性能（有关临时表的内容参阅第4章）。不像临时表，衍生表不需要预先填充固定数据。

这个示例演示了如何在SELECT语句的FROM子句中使用衍生表：

```
SELECT DISTINCT s.PurchaseOrderNumber
FROM Sales.SalesOrderHeader s
INNER JOIN (SELECT SalesOrderID
        FROM Sales.SalesOrderDetail
        WHERE UnitPrice BETWEEN 1000 AND 2000) d ON
s.SalesOrderID = d.SalesOrderID
```

这个查询返回如下的部分结果：

```
PurchaseOrderNumber
PO8410140860
PO12325137381
PO1160166903
PO1073122178
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
...
PO15486173227
PO14268145224

(1989行受影响)
```

解析

这个示例的查询搜索Sales.SalesOrderHeader表中的PurchaseOrderNumber，以查询任何包含UnitPrice从1000到2000的产品的订单。

这个查询使用INNER JOIN将表联结到一个衍生表。衍生表查询包含在括号中，且后面跟着一个表别名。衍生表本身也是一个查询，并且不需要使用临时表来保存结果。因为消除了SQL Server在使用前创建和分配临时表所需要的步骤，所以使用衍生表的查询有时比临时表性能好很多。

1.8.6 使用 UNION 组合结果集

使用UNION运算符可以追加两个或多个SELECT语句的结果到一个结果集中。每一个合并的SELECT语句必须有相同数量的列，列次序相同并且有相同的或可兼容的数据类型，如下例所示：


```
SELECT   BusinessEntityID, GETDATE() QuotaDate, SalesQuota
FROM      Sales.SalesPerson
WHERE      SalesQuota > 0
      UNION
SELECT   BusinessEntityID, QuotaDate, SalesQuota
FROM      Sales.SalesPersonQuotaHistory
WHERE      SalesQuota > 0
ORDER BY BusinessEntityID DESC, QuotaDate DESC
```

这个查询返回如下的（部分）结果：

```
SalesPersonID   QuotaDate                SalesQuota
290             2007-09-01 14:26:28.870  250000.00
290             2004-04-01 00:00:00.000  421000.00
290             2004-01-01 00:00:00.000  399000.00
290             2003-10-01 00:00:00.000  389000.00
...
268             2001-10-01 00:00:00.000  7000.00
268             2001-07-01 00:00:00.000  28000.00

(177行受影响)
```

解析

这个查询把两个结果集追加到一个结果集中。第一个结果集返回BusinessEntityID、当前的日期函数（更多信息参阅第8章）和SalesQuota。因为GETDATE()是一个函数，它本身不会返回一个列名，所以在这里使用了QuotaDate作为列别名：

```
SELECT   BusinessEntityID, GETDATE() QuotaDate, SalesQuota
FROM      Sales.SalesPerson
```

WHERE子句过滤出那些SalesQuota大于0的销售人员的数据：

```
WHERE      SalesQuota > 0
```

查询的下一部分是UNION运算符，它通过第二个查询来追加不重复的结果：

```
UNION
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第二个查询从Sales.SalesPersonQuotaHistory提取数据，该表保存了销售人员的销售定额的历史：

```
SELECT   BusinessEntityID, QuotaDate, SalesQuota
FROM      Sales.SalesPersonQuotaHistory
WHERE     SalesQuota > 0
```

ORDER BY子句根据BusinessEntityID和QuotaDate以逆序排列结果集。如果需要ORDER BY的话就必须把它放在查询的最后，并且不能是最后那个UNION的查询之前的查询。ORDER BY子句应该只引用第一个结果集中的列名：

```
ORDER BY BusinessEntityID DESC, QuotaDate DESC
```

再来看看这个结果，对于一个销售人员，你可以看到QuotaDate为'2005-02-27'的记录排列在最前面。这是由GETDATE()函数得到的日期。其余那些SalesPersonID为290的行都来自Sales.SalesPerson-
34 QuotaHistory表：

```
SalesPersonID   QuotaDate                SalesQuota
290             2005-02-27 10:10:12.587  250000.00
290             2004-04-01 00:00:00.000  421000.00
290             2004-01-01 00:00:00.000  399000.00
290             2003-10-01 00:00:00.000  389000.00
```

要记住，默认情况下使用UNION运算符会移除所有重复行，并且基于第一个结果集显示列名。对于大结果集来说，这可能会是一个相当耗费资源的操作，所以如果你不需要去除重复数据或者数据本身就是不重复的话，可以在UNION中使用ALL关键字：

```
UNION ALL
```

加了ALL子句之后，重复的行将不会被移除。

警告 与使用DISTINCT相似，使用UNION而不用UNION ALL会带来额外的查询资源开销。如果不想移除重复的行，就使用UNION ALL。

1.9 使用 APPLY 来为每行调用表值函数

APPLY用于为外部查询的每行调用表值函数。表值函数根据一个或多个参数返回结果集。使用APPLY，这些参数的输入就可以是引用的左表的列。如果左表包含必须由表值函数进行计算的列或行，并且这个函数的结果要附加到左表中，那么使用APPLY会很有用。

CROSS APPLY和INNER JOIN差不多，左表和表值函数不匹配的行不会出现在结果集中。OUTER APPLY和OUTER JOIN差不多，不匹配的行仍然会使用函数结果中的NULL值在结果集中返回。

下面两个技巧会演示CROSS APPLY和OUTER APPLY。

注解 下面的示例涉及了FROM和JOIN的示例以及用户定义的表值函数。有关表值函数的更多细节可以参阅第11章。

1.9.1 使用 CROSS APPLY

在这个示例中，我们创建了一个表值函数，它会根据传入的WorkOrderID返回工作订单的路由信息：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE FUNCTION dbo.fn_WorkOrderRouting
  (@WorkOrderID int) RETURNS TABLE
AS
RETURN
  SELECT WorkOrderID,
       ProductID,
       OperationSequence,
       LocationID
   FROM Production.WorkOrderRouting
   WHERE WorkOrderID = @WorkOrderID
GO
```


接着，把Production.WorkOrder表中的WorkOrderID传入新函数：

```
SELECT   w.WorkOrderID,
      w.OrderQty,
      r.ProductID,
      r.OperationSequence
FROM Production.WorkOrder w
  CROSS APPLY dbo.fn_WorkOrderRouting
   (w.WorkOrderID) AS r
ORDER BY   w.WorkOrderID,
         w.OrderQty,
         r.ProductID
```

这个查询返回下面的（部分）结果：

```
WorkOrderID   OrderQty    ProductID   OperationSequence
13            4           747         1
13            4           747         2
13            4           747         3
13            4           747         4
13            4           747         6
...
72586         1           803         6
72587         19          804         1
72587         19          804         6

(67131行受影响)
```

解析

这个技巧的第一部分用于创建表值函数。函数接受一个参数@WorkOrderID，执行后会根据某个WorkOrderID从Production.WorkOrderRouting表返回WorkOrderID、ProductID、OperationSequence和LocationID。

示例的下一个查询从Production.WorkOrder表返回WorkOrderID和OrderQty。除此之外还有表值函数中的两列：

```
SELECT   w.WorkOrderID,
      w.OrderQty,
      r.ProductID,
      r.OperationSequence
```

接下来就是这个技巧的关键部分了。注意到在FROM子句中，使用了CROSS APPLY把Production.WorkOrder表联结到一个新的表值函数。唯一和JOIN子句不同的是，在联结条件后没有跟随ON，而是在函数名后的圆括号中，把左表Production.WorkOrder的w.WorkOrderID传入到了表值函数中：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FROM Production.WorkOrder w
  CROSS APPLY dbo.fn_WorkOrderRouting
  (w.WorkOrderID) AS r
```

36 这个函数就像一个普通表那样使用了别名，名字为“r”。

最后对结果进行排序：

```
ORDER BY   w.WorkOrderID,
           w.OrderQty,
           r.ProductID
```

对于结果中WorkOrderID为13的记录，在调用表的WorkOrderID和OrderQty之后，每一个和WorkOrderRouting关联的行都返回了。对于WorkOrder表的每一行，都会根据从fn_WorkOrderRouting匹配WorkOrderID返回的每一行进行重复。

1.9.2 使用 OUTER APPLY

要演示OUTER APPLY，需要向Production.WorkOrder表插入一个新行（要了解INSERT命令，请参阅第2章）：

```
INSERT INTO [AdventureWorks].[Production].[WorkOrder]
           ([ProductID]
           ,[OrderQty]
           ,[ScrappedQty]
           ,[StartDate]
           ,[EndDate]
           ,[DueDate]
           ,[ScrapReasonID]
           ,[ModifiedDate])
     VALUES
           (1,
           1,
           1,
           GETDATE(),
           GETDATE(),
           GETDATE(),
           1,
           GETDATE())
```

因为这是一个新行，而且Production.WorkOrder有一个IDENTITY列WorkOrderID，所以新行在表中的WorkOrderID最大。同样，由于是刚加入的行，新行在Production.WorkOrderRouting表中没有关联的值。

然后，执行CROSS APPLY查询，这次限定它只返回新插入行的数据：

```
SELECT   w.WorkOrderID,
       w.OrderQty,
       r.ProductID,
       r.OperationSequence
FROM Production.WorkOrder AS w
  CROSS APPLY dbo.fn_WorkOrderRouting
    (w.WorkOrderID) AS r
WHERE w.WorkOrderID IN
       (SELECT MAX(WorkOrderID)
        FROM Production.WorkOrder)
```

这个查询什么都没有返回，因为左表的新行没有匹配：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
WorkOrderID   OrderQty   ProductID   OperationSequence

(0行受影响)
```


现在，再来试试OUTER APPLY，尽管在表值函数中没有关联的值，它依然会从WorkOrder返回行：

```
SELECT   w.WorkOrderID,
      w.OrderQty,
      r.ProductID,
      r.OperationSequence
FROM Production.WorkOrder AS w
  OUTER APPLY dbo.fn_WorkOrderRouting
   (w.WorkOrderID) AS r
WHERE w.WorkOrderID IN
      (SELECT MAX(WorkOrderID)
       FROM Production.WorkOrder)
```

这个查询返回：

```
WorkOrderID   OrderQty   ProductID   OperationSequence
72592         1          NULL        NULL

(1行受影响)
```

解析

CROSS APPLY和OUTER APPLY提供了使用表值函数查找列的新方法。 在演示中，我们CROSS APPLY一个在表值函数中没有匹配的行。CROSS APPLY和INNER JOIN差不多，没有返回行。在示例的第二个查询中，使用了OUTER APPLY，它和OUTER JOIN相似，这次对于表值函数没有匹配的行返回NULL。

1.10 数据源高级技术

接下来的一组技巧会介绍用于取样、操作和比较数据源（数据源是FROM子句中的任何有效的数据源引用）的高级技术，包括如下内容：

- 使用TABLESAMPLE返回样本行；
- 使用PIVOT把值转化为列，并且使用聚合根据新的列来分组数据；
- 使用UNPIVOT来规范化重复的列分组；
- 使用INTERSECT和EXCEPT操作数来返回仅在左查询中的不重复行（使用EXCEPT），或者左查询和右查询中都不重复的行（使用INTERSECT）。

1.10.1 使用 TABLESAMPLE 来返回随机行

使用TABLESAMPLE，我们可以从From子句的表中提取一些样本行。这个取样可以基于一定百分比的行。当应用程序只需要一些样本行而不是完整的结果集时，可以使用TABLESAMPLE来实现。

这个示例演示了使用TABLESAMPLE的一个查询，它从某个数据源返回一定百分比的随机行：

```
SELECT FirstName,LastName
FROM Person.Person
TABLESAMPLE SYSTEM (2 PERCENT)
```

这个查询返回下面的（部分）结果：

```
FirstName    LastName
Andre        Suri
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Adam        Turner
Eric        Turner
Jackson     Turner
Meghan      Rowe
...
(232行受影响)
```

再次执行又返回了新的一组结果（部分）：

```
FirstName   LastName
Robert      King
Ricardo     Raje
Jose        King
Ricardo     Chande
...
Martin      Perez
Carlos      Collins

(198行受影响)
```

解析

TABLESAMPLE用于从查询结果集中提取一些样本行。在这个示例中，我们从Person.Person表取样2%的行。然而，不要让这个“百分比”唬住，这个百分比是表数据页的百分比。一旦选中了一个样本页，那么页中的所有行都会返回。由于页的填充状态各不相同，所以返回行的数量也不一样。你会发现示例的第一次查询执行后返回232行，而第二次返回198行。其实就算指定了行数，SQL Server也会把它转化为百分比，然后用同样的方法找出一定百分比的数据页。

1.10.2　使用 PIVOT 把单列值转化为多列和聚合数据

PIVOT运算符能让我们创建交叉表的查询，它把值转化为多列，使用聚合来根据新列对数据进行分组。

PIVOT使用如下的语法：

```
FROM  table_source
PIVOT     ( aggregate_function ( value_column )
          FOR pivot_column
          IN ( <column_list>)
    ) table_alias
```

39

表1-3描述了PIVOT的参数。

表1-3　PIVOT参数

参　数	描　述
table_source	要旋转数据的表
aggregate_function(value_column)	要在某列上使用的集合函数
pivot_column	用于创建列头的列
column_list	要从旋转列中旋转的值
table_alias	旋转结果集的表别名

下一个示例演示如何实现类似微软Excel旋转表特性的PIVOT和聚合数据功能：把一列中的值转到多列，在结果中显示聚合数据。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

示例的第一部分显示旋转之前的数据。查询结果显示了雇员的轮换和他们所在的部门：

```
SELECT   s.Name ShiftName,
      h.BusinessEntityID,
      d.Name DepartmentName
FROM   HumanResources.EmployeeDepartmentHistory h
INNER JOIN HumanResources.Department d ON
      h.DepartmentID = d.DepartmentID
INNER JOIN HumanResources.Shift s ON
      h.ShiftID = s.ShiftID
WHERE   EndDate IS NULL AND
      d.Name IN ('Production', 'Engineering', 'Marketing')
ORDER BY ShiftName
```

注意到，在同一个列中列出了各种部门：

```
ShiftName    BusinessEntityID    DepartmentName
Day          3                   Engineering
Day          9                   Engineering
...
Day          2                   Marketing
Day          6                   Marketing
...
Evening      25                  Production
Evening      18                  Production
Night        14                  Production
Night        27                  Production
...
Night        252                 Production

(194行受影响)
```

40

下面的查询把轮换的雇员数量和部门值旋转到列中：

```
SELECT   ShiftName,
      Production,
      Engineering,
      Marketing
FROM
(SELECT   s.Name ShiftName,
      h.BusinessEntityID,
      d.Name DepartmentName
FROM   HumanResources.EmployeeDepartmentHistory h
INNER JOIN HumanResources.Department d ON
      h.DepartmentID = d.DepartmentID
INNER JOIN HumanResources.Shift s ON
      h.ShiftID = s.ShiftID
WHERE   EndDate IS NULL AND
      d.Name IN ('Production', 'Engineering', 'Marketing')) AS a
PIVOT
(
     COUNT(BusinessEntityID)
     FOR DepartmentName IN ([Production], [Engineering], [Marketing])
) AS b
ORDER BY ShiftName
```

这个查询返回：

```
ShiftName    Production    Engineering    Marketing
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Day         79          6           9
Evening     54          0           0
Night       46          0           0

(3行受影响)
```

解析

PIVOT查询的结果返回了不同轮换时间和部门的雇员计数。查询先列出要返回的字段名：

```
SELECT   ShiftName,
      Production,
      Engineering,
      Marketing
```

注意到这些字段本来是转换后的行，但最后变成了列名。

FROM子句引用了子查询（本例开始时的那个查询。子查询用任意一个名字“a”作为别名）：

```
FROM
(SELECT   s.Name ShiftName,
      h. BusinessEntityID,
      d.Name DepartmentName
FROM   HumanResources.EmployeeDepartmentHistory h
INNER JOIN HumanResources.Department d ON
      h.DepartmentID = d.DepartmentID
INNER JOIN HumanResources.Shift s ON
      h.ShiftID = s.ShiftID
WHERE    EndDate IS NULL AND
      d.Name IN ('Production', 'Engineering', 'Marketing')) AS a
```

41

在圆括号中，查询指定了哪些列要被聚合（以及如何聚合）。在本例中，要计算雇员的数量：

```
PIVOT
(COUNT(BusinessEntityID)
```

在聚合部分之后，FOR语句决定了哪些行值需要被转化成列。和普通IN子句不同的是，在字符串周围不用单引号，而是使用方括号。DepartmentName就是其值需要转化成旋转后的列的那个数据列：

```
FOR DepartmentName IN ([Production], [Engineering], [Marketing]))
```

注解　*要旋转的列名称不能存在于将要旋转的基础表或视图的查询列中。*

最后，用一个关闭的圆括号关闭PIVOT操作。然后把PIVOT操作像表一样起个别名，在这里使用名字b：

```
AS b
```

然后根据ShiftName排列结果：

```
ORDER BY ShiftName
```

结果显示了PIVOT操作中由FOR部分固定的3个列和根据ShiftName聚合的雇员数。

1.10.3　使用 UNPIVOT 规范化数据

UNPIVOT命令的功能几乎和PIVOT相反，它把列转成行。它和PIVOT有相同的语法，不同的是指定UNPIVOT。

这个示例演示如何使用UNPIVOT移除经常在非规范化表中看到的一组重复列。示例的第一部分创建了一个非规范化表，表中有几个重复的、递增的电话号码列：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE TABLE dbo.Contact
   (EmployeeID int NOT NULL,
    PhoneNumber1 bigint,
    PhoneNumber2 bigint,
    PhoneNumber3 bigint)
GO

INSERT dbo.Contact
(EmployeeID, PhoneNumber1, PhoneNumber2, PhoneNumber3)
VALUES( 1, 2718353881, 3385531980, 5324571342)

INSERT dbo.Contact
(EmployeeID, PhoneNumber1, PhoneNumber2, PhoneNumber3)
VALUES( 2, 6007163571, 6875099415, 7756620787)

INSERT dbo.Contact
(EmployeeID, PhoneNumber1, PhoneNumber2, PhoneNumber3)
VALUES( 3, 9439250939, NULL, NULL)
```

现在使用UNPIVOT把重复的电话号码转化成更规范化的形式（重新利用一个PhoneValue字段来替代重复电话号码列多次）：

```
SELECT   EmployeeID,
      PhoneType,
      PhoneValue
FROM
(SELECT   EmployeeID, PhoneNumber1, PhoneNumber2, PhoneNumber3
FROM dbo.Contact) c
UNPIVOT
   (PhoneValue FOR PhoneType IN ([PhoneNumber1], [PhoneNumber2], [PhoneNumber3])
) AS p
```


这个查询返回：

```
EmployeeID   PhoneType      PhoneValue
1            PhoneNumber1   2718353881
1            PhoneNumber2   3385531980
1            PhoneNumber3   5324571342
2            PhoneNumber1   6007163571
2            PhoneNumber2   6875099415
2            PhoneNumber3   7756620787
3            PhoneNumber1   9439250939

(7行受影响)
```

解析

UNPIVOT的示例从选择3个列开始。EmployeeID从子查询中得到。其他两列，PhoneType和PhoneValue，都在之后的UNPIVOT语句中定义：

```
SELECT   EmployeeID,
      PhoneType,
      PhoneValue
```

然后在FROM子句中引用一个子查询。子查询选择了联系表中的所有4列。表使用了字母c作为别名（表别名是随便起的）：

```
FROM
(SELECT    EmployeeID, PhoneNumber1, PhoneNumber2, PhoneNumber3
FROM dbo.Contact) c
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

一个叫PhoneValue的新列（在SELECT中引用的）保存了3个非规范化电话号码列中的各个电话号码：

```
UNPIVOT
(PhoneValue FOR PhoneType IN ([PhoneNumber1], [PhoneNumber2], [PhoneNumber3])
```

FOR引用了旋转列的名字——PhoneType，它保存非规范化表的列名。PhoneType后面的IN子句中列出了原始表中的列，这些列要被收缩到一个列中。

最后，使用关闭的圆括号，然后给它随便起一个别名，在这里是p：

```
) AS p
```

43 这个查询返回了合并到两列中的电话数据，一列用于描述电话类型，而另外一列保存了实际的电话号码。我们还注意到，查询返回了7行而不是9行。这是因为对于EmployeeID等于3的记录，只有非NULL的值才会返回。UNPIVOT不返回旋转后的结果集中的NULL值。

1.10.4 使用 EXCEPT 和 INTERSECT 返回不重复的或匹配的行

使用INTERSECT和EXCEPT运算符，可以返回只存在于左查询中的不重复行（使用EXCEPT），或者同时存在于左右两个查询中的不重复行（使用INTERSECT）。

INTERSECT和EXCEPT在数据集比较时很有用。例如，如果你需要比较测试表和正式表中的行，可以使用EXCEPT轻松找到和获取那些在一个表中存在而另外一个表中不存在的行。这些操作数在数据恢复的时候同样有用，因为你可以从过去的数据库中恢复丢失的数据，先和当前正式表中的数据进行比较，然后恢复相应的被删除的行。

在这个示例中，我们创建了演示表，它包含了Production.Product表中的部分数据：

```
-- 先创建两个基于Production.Product的表，用于演示EXCEPT和INTERSECT
-- 更多有关ROW_NUMBER的信息参阅第8章

-- 创建TableA
SELECT   prod.ProductID,
      prod.Name
INTO dbo.TableA
FROM
(SELECT ProductID,
      Name,
      ROW_NUMBER() OVER (ORDER BY ProductID) RowNum
FROM Production.Product) prod
WHERE RowNum BETWEEN 1 and 20

-- 创建TableB
SELECT   prod.ProductID,
      prod.Name
INTO dbo.TableB
FROM
(SELECT ProductID,
      Name,
      ROW_NUMBER() OVER (ORDER BY ProductID) RowNum
FROM Production.Product) prod
WHERE RowNum BETWEEN 10 and 29
```

这个查询返回：

```
(20行受影响)
(20行受影响)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

现在使用EXCEPT运算符检测只在查询左表TableA中存在而不在TableB中存在的行：

```
SELECT   ProductID,
         Name
FROM     dbo.TableA
EXCEPT
SELECT   ProductID,
         Name
FROM     dbo.TableB
```

这个查询返回：

```
ProductID   Name
1           Adjustable Race
2           Bearing Ball
3           BB Ball Bearing
4           Headset Ball Bearings
316         Blade
317         LL Crankarm
318         ML Crankarm
319         HL Crankarm
320         Chainring Bolts

(9行受影响)
```

要显示在两个结果集中都能找到的不重复值，可以使用INTERSECT运算符：

```
SELECT   ProductID,
         Name
FROM     dbo.TableA
INTERSECT
SELECT   ProductID,
         Name
FROM     dbo.TableB
```

这个查询返回：

```
ProductID   Name
321         Chainring Nut
322         Chainring
323         Crown Race
324         Chain Stays
325         Decal 1
326         Decal 2
327         Down Tube
328         Mountain End Caps
329         Road End Caps
330         Touring End Caps
331         Fork End

(11行受影响)
```

解析

这个示例先创建了两个表（使用INTO），它们包含了重叠的一些行。

注解 关于在技巧中填充表时使用的ROW_NUMBER函数的更多细节请参阅第8章。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第一个表TableA包含了Production.Product表的前20行（根据ProductID排序）。第二个表TableB同样接受了另外20行，有一半与TableA的行重叠。

要找到哪些行仅存在于TableA，可以在第一个查询的FROM子句之后、第二个查询之前使用EXCEPT运算符：

```
SELECT   ProductID,
         Name
FROM     dbo.TableA
EXCEPT
SELECT   ProductID,
         Name
FROM     dbo.TableB
```

由于使用了EXCEPT，两个查询必须有相同数量的列。那些列也需要有兼容的数据类型（每个查询中的列名不需要一致）。EXCEPT的优势在于它会计算所有列以检测是否存在匹配，而且它比INNER JOIN（需要显式地在每个列上对两个数据源的表进行联结）效率更高。

EXCEPT查询的结果显示了TableA的前9行，它们都没有包含在TableB中。

在第二个示例中，使用INTERSECT来显示两个表中重叠的行。和EXCEPT差不多，INTERSECT放在两个查询中间：

```
SELECT   ProductID,
         Name
FROM     dbo.TableA
INTERSECT
SELECT   ProductID,
         Name
FROM     dbo.TableB
```

这个查询返回了两个表中重叠的11行。INTERSECT和EXCEPT一样也有兼容数据类型和列数限制规则。

1.11 汇总数据

在之后的3个技巧中，我会演示如何使用下面的运算符来汇总结果集中的数据：

- 使用CUBE根据GROUP BY子句中的列来为结果集增加汇总值；
- 使用ROLLUP和GROUP BY，根据GROUP BY子句中列的次序增加分层的数据汇总；
- 使用GROUPING SETS运算符在一个结果集中进行自定义聚合，而不需要使用UNION ALL。

下面将先演示如何使用CUBE汇总数据。

1.11.1 使用 CUBE 汇总数据

46

CUBE为结果集增加行，根据GROUP BY子句中的列来汇总总数值。

这个示例演示了一个根据产品放置的货架来返回产品总数量的查询：

```
SELECT   i.Shelf,
         SUM(i.Quantity) Total
FROM     Production.ProductInventory i
GROUP BY CUBE (i.Shelf)
```

这个查询返回了如下的（部分）结果：

```
Shelf    Total
A        26833
B        12672
C        19868
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
D         17353
E         31979
F         21249
G         40195
H         20055
J         12154
K         16311
L         13553
M         3567
N         5254
N/A       30582
R         23123
S         5912
T         10634
U         18700
V         2635
W         2908
Y         437
NULL      335974

(22行受影响)
```

在下一个查询中，我们将LocationID增加到SELECT和GROUP BY子句中：

```
SELECT   i.Shelf,
   i.LocationID,
   SUM(i.Quantity) Total
FROM   Production.ProductInventory i
GROUP BY CUBE (i.Shelf,i.LocationID)
```

查询返回几个级别的总和，首先是根据（部分）地区：

```
Shelf     LocationID     Total
A         1              2727
C         1              13777
D         1              6551
...
K         1              6751
L         1              7537
NULL      1              72899
```


在同一结果集中，在后面你也会看到根据货架统计的总和，最后面是所有货架和地区的产品总和：

```
Shelf     LocationID     Total
...
T         NULL           10634
U         NULL           18700
V         NULL           2635
W         NULL           2908
Y         NULL           437
NULL      NULL           335974
```

解析

因为第一个查询根据货架分组，并且使用了GROUP BY CUBE，所以在结果集的底部增加了一行，显示所有货架上的产品总和：

```
GROUP BY CUBE (i.Shelf)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

警告 当使用CUBE时，小心不要重复计算聚合值。

它的语法与SQL Server早期的版本不同。在SQL Server 2008中，CUBE是跟在GROUP BY之后的，而不是以WITH CUBE方式出现在GROUP BY之后。还要注意圆括号包含的列的列表：

```
GROUP BY CUBE (i.Shelf,i.LocationID)
```

为查询增加额外的列，将它包含在GROUP BY CUBE子句中，你会看到每个分组组合的聚合值。CUBE通常用于生成报表，它提供了一种为分组列求和的简单方法。

注解 在SQL Server早期的版本中，你可能用过COMPUTE BY，它为查询提供了相似的聚合功能。为了向后兼容，微软不推荐使用此功能。和CUBE不同，COMPUTE BY在原始查询结果后再创建一个全新的汇总结果集，这种方式往往增加了调用的应用程序处理的难度。

1.11.2 使用 ROLLUP 来汇总数据

GROUP BY ROLLUP用来根据GROUP BY子句中列的次序来增加层次化的数据汇总。

这个示例获取了货架、产品名和产品的总数：

```
SELECT   i.Shelf,
      p.Name,
      SUM(i.Quantity) Total
FROM   Production.ProductInventory i
INNER JOIN Production.Product p ON
   i.ProductID = p.ProductID
GROUP BY ROLLUP (i.Shelf, p.Name)
```

48

这个查询返回如下的（部分）结果：

```
Shelf   Name                 Total
A       Adjustable Race      761
A       BB Ball Bearing      909
...
A       NULL                 26833
B       Adjustable Race      324
B       BB Ball Bearing      443
B       Bearing Ball         318
...
B       Touring Front Wheel  304
B       NULL                 12672
C       hain                 236
C       Chain Stays          585
Y       LL Spindle/Axle      209
Y       NULL                 437
NULL    NULL                 335974
```

解析

GROUP BY ROLLUP中列的次序会对如何聚合数据产生影响。该查询中的ROLLUP聚合了每一个货架上每次变动的总数量。注意那个Shelf为A而Name为NULL的行，它保存了A货架上货物的总数量。同样注意到，最后那行是所有产品数量的总和。CUBE创建了聚合所有选择列组合出来的结果集，而ROLLUP生成层次化的值的聚合。

```
GROUP BY ROLLUP (i.Shelf, p.Name)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

ROLLUP聚合出总数量和以货架为单位的总计。总计没有生成货物名称，而我在CUBE中指定了。

就像CUBE一样，ROLLUP与SQL Server之前版本的语法稍有不同。在SQL Server 2008中，ROLLUP跟在GROUP BY之后，而不是在GROUP BY子句后用WITH ROLLUP。也要注意包含在括号中的列。

1.11.3 使用分组集创建自定义汇总

SQL Server 2008引入了在单个查询结果集中自定义分组集的能力，而不再是只能借助多个UNION ALL来实现。而且与前面演示的CUBE和ROLLUP操作相比，使用分组集可以更好地控制聚合内容。这一操作使用GROUPING SETS来完成。

首先，我会演示为查询定义示例业务需求。假设我希望得到一个包含三个不同聚合数量汇总的结果集。具体来说，我希望查看每个货架上的产品数量、每个货架上每种产品的数量以及每个地区每种产品的数量。

在SQL Server的早期版本中，要达到这一点必须使用UNION ALL：

```
SELECT
     NULL,
     i.LocationID,
     p.Name,
     SUM(i.Quantity) Total
FROM   Production.ProductInventory i
INNER JOIN Production.Product p ON
   i.ProductID = p.ProductID
WHERE Shelf IN ('A','C') AND
   Name IN ('Chain', 'Decal', 'Head Tube')
GROUP BY i.LocationID, p.Name
UNION ALL
SELECT
     i.Shelf,
        NULL,
        NULL,
     SUM(i.Quantity) Total
FROM   Production.ProductInventory i
INNER JOIN Production.Product p ON
   i.ProductID = p.ProductID
WHERE Shelf IN ('A','C') AND
   Name IN ('Chain', 'Decal', 'Head Tube')
GROUP BY i.Shelf
UNION ALL
SELECT
     i.Shelf,
     NULL,
     p.Name,
     SUM(i.Quantity) Total
FROM   Production.ProductInventory i
INNER JOIN Production.Product p ON
   i.ProductID = p.ProductID
WHERE Shelf IN ('A','C') AND
   Name IN ('Chain', 'Decal', 'Head Tube')
GROUP BY i.Shelf, p.Name
```

查询返回：

```
LocationID   Name   Total
NULL         1      Chain       236
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
NULL        5     Chain       192
NULL        50    Chain       161
NULL        20    Head Tube   544
A           NULL  NULL        897
C           NULL  NULL        236
A           NULL  Chain       353
C           NULL  Chain       236
A           NULL  Head Tube   544

(9行受影响)
```

在SQL Server 2008中，可以使用GROUPING SETS操作符替代在单个结果集中定义你想要返回的各种聚合运算，从而节省额外的代码。

```
SELECT
      i.Shelf,
      i.LocationID,
      p.Name,
      SUM(i.Quantity) Total
FROM   Production.ProductInventory i
INNER JOIN Production.Product p ON
   i.ProductID = p.ProductID
WHERE Shelf IN ('A','C') AND
   Name IN ('Chain', 'Decal', 'Head Tube')
GROUP BY GROUPING SETS
   ((i.Shelf), (i.Shelf, p.Name), (i.LocationID, p.Name))
```


该查询返回与前面查询相同的结果集（只是排序有一点不同）：

```
Shelf   LocationID  Name        Total
NULL    1           Chain       236
NULL    5           Chain       192
NULL    50          Chain       161
NULL    20          Head Tube   544
A       NULL        Chain       353
A       NULL        Head Tube   544
A       NULL        NULL        897
C       NULL        Chain       236
C       NULL        NULL        236

(9行受影响)
```

解析

新的GROUPING SETS操作符允许在单个查询中定义不同的聚合分组，避免了使用UNION ALL将多个查询附加到一起。这个技巧的示例的关键是以下两行代码：

```
GROUP BY GROUPING SETS
   ((i.Shelf), (i.Shelf, p.Name), (i.LocationID, p.Name))
```

要注意的是，不像常规的聚合查询，GROUP BY子句后面不是一些列名，而是GROUPING SETS。GROUPING SETS之后是圆括号和列名的分组，每一个分组都括在圆括号中。

1.11.4 展现GROUPING生成的行

你可能会注意到，在前面的技巧中，进行分组的那些行在没有参与聚合总和的列中有NULL值。例如，当为货架C生成总和时，地区和产品名称列为NULL：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
C    NULL    NULL    236
```

如果数据中确定不包含NULL，那么NULL值是可接受的；但是，如果存在NULL值呢？如何区分出那些“存储的”NULL和由ROLLUP、CUBE和GROUPING SETS生成的NULL呢？

为了确定这个问题，可以使用GROUPING和GROUPING_ID函数。我会在这个技巧中讨论GROUPING，然后在下一个技巧中讨论GROUPING_ID。在SQL Server的早期版本中，GROUPING是可用的，它可以简单地评估一行是否为聚合的产物。例如，下面的查询使用CASE语句来评估行是否为根据货架统计的总和、根据地区统计的总和、全部合计或是通常的非cube的行：

```
SELECT
   i.Shelf,
   i.LocationID,
   CASE
     WHEN GROUPING(i.Shelf) = 0 AND
          GROUPING(i.LocationID) = 1 THEN 'Shelf Total'
     WHEN GROUPING(i.Shelf) = 1 AND
          GROUPING(i.LocationID) = 0 THEN 'Location Total'
     WHEN GROUPING(i.Shelf) = 1 AND
          GROUPING(i.LocationID) = 1 THEN 'Grand Total'
      ELSE 'Regular Row'
   END RowType,
   SUM(i.Quantity) Total
FROM   Production.ProductInventory i
WHERE LocationID = 2
GROUP BY CUBE (i.Shelf,i.LocationID)
```

查询返回：

```
Shelf    LocationID    RowType          Total
B        2             Regular Row      900
C        2             Regular Row      1557
D        2             Regular Row      3092
NULL     2             Location Total   5549
NULL     NULL          Grand Total      5549
B        NULL          Shelf Total      900
C        NULL          Shelf Total      1557
D        NULL          Shelf Total      3092
(8行受影响)
```

解析

GROUPING函数允许你区分并操作那些使用CUBE、ROLLUP和GROUPING SETS聚合自动生成的行。在这个示例中，先用平常的SELECT语句，该语句包含了Shelf和Location列：

```
SELECT
   i.Shelf,
   i.LocationID,
```

在这之后，是一个CASE语句，用来计算GROUPING语句的返回值的合并。

提示 CASE的更多内容参阅第9章。

如果GROUPING返回值为1（真），表示NULL不是实际的数据值，而是聚合操作的结果，可以理解为值“全部”。因此在示例中，如果由于CUBE聚合过程而不是数据本身，造成货架的值不为NULL但地区ID

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

为NULL，则返回字符串Shelf Total：

```
CASE
  WHEN GROUPING(i.Shelf) = 0 AND
       GROUPING(i.LocationID) = 1 THEN 'Shelf Total'
```

然后以相似的逻辑继续，只是这一次由于CUBE聚合过程造成货架的值为NULL，但地区不为NULL，则生成地区总和：

52

```
  WHEN GROUPING(i.Shelf) = 1 AND
       GROUPING(i.LocationID) = 0 THEN 'Location Total'
```

最后的WHEN定义了由于CUBE聚合过程同时造成货架值与地区值都为NULL时，表示行包含了结果集的全部合计：

```
  WHEN GROUPING(i.Shelf) = 1 AND
       GROUPING(i.LocationID) = 1 THEN 'Grand Total'
```

GROUPING只返回1或0，但在SQL Server 2008中，也可以选择使用GROUPING_ID以细粒度来计算分组，在下一个技巧中我将演示它。

1.11.5 使用 GROUPING_ID 标识分组级别

注解 这个技巧假设你理解二进制数字系统。

随着增加到GROUP BY的新列和可分组与聚合的唯一数据值，标识出哪些行属于哪一种聚合变得更加困难。例如，假设我有一个非聚合报表，显示了地区为3和箱柜为1和2的产品数量：

```
SELECT
   i.Shelf,
   i.LocationID,
   i.Bin,
   i.Quantity
FROM   Production.ProductInventory i
WHERE i.LocationID IN (3) AND
      i.Bin IN (1,2)
```

这个查询只返回了两行：

```
Shelf   LocationID    Bin     Quantity
A       3             2       41
A       3             1       49
```

现在如果希望生成基于货架、地区和箱柜的各种组合的聚合报表，可以使用CUBE产生所有这些潜在组合的汇总：

```
SELECT
   i.Shelf,
   i.LocationID,
   i.Bin,
   SUM(i.Quantity) Total
FROM   Production.ProductInventory i
WHERE i.LocationID IN (3) AND
      i.Bin IN (1,2)
GROUP BY CUBE (i.Shelf,i.LocationID, i.Bin)
ORDER BY i.Shelf, i.LocationID, i.Bin
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

尽管查询返回了CUBE期望的各种聚合，但结果仍然是难以理解的：

```
Shelf   LocationID   Bin    Total
NULL    NULL         NULL   90
NULL    NULL         1      49
NULL    NULL         2      41
NULL    3            NULL   90
NULL    3            1      49
NULL    3            2      41
A       NULL         NULL   90
A       NULL         1      49
A       NULL         2      41
A       3            NULL   90
A       3            1      49
A       3            2      41

(12行受到影响)
```

这时GROUPING_ID就派上用场了。使用这个函数，可以确定行的分组级别。这个函数比GROUPING更加复杂，这是因为GROUPING_ID用一个或多个列作为输入，然后返回列的二进制数字计算结果对应的整数。

这可以通过示例进行最好的描述，因此我将使用前面的查询，并增加CASE逻辑来返回合适的行描述符：

```
SELECT
   i.Shelf,
   i.LocationID,
   i.Bin,
   CASE GROUPING_ID(i.Shelf,i.LocationID, i.Bin)
      WHEN 1 THEN 'Shelf/Location Total'
      WHEN 2 THEN 'Shelf/Bin Total'
      WHEN 3 THEN 'Shelf Total'
      WHEN 4 THEN 'Location/Bin Total'
      WHEN 5 THEN 'Location Total'
      WHEN 6 THEN 'Bin Total'
      WHEN 7 THEN 'Grand Total'
    ELSE 'Regular Row'
   END,
   SUM(i.Quantity) Total
FROM   Production.ProductInventory i
WHERE i.LocationID IN (3) AND
      i.Bin IN (1,2)
GROUP BY CUBE (i.Shelf,i.LocationID, i.Bin)
ORDER BY i.Shelf, i.LocationID, i.Bin
```

我会在“解析”部分中解释每个整数的意思。这个查询返回的结果体现了CUBE产生的各种聚合结果：

```
Shelf   LocationID   Bin    Regular Row          Total
NULL    NULL         NULL   Grand Total          90
NULL    NULL         1      Bin Total            49
NULL    NULL         2      Bin Total            41
NULL    3            NULL   Location Total       90
NULL    3            1      Location/Bin Total   49
NULL    3            2      Location/Bin Total   41
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
A       NULL        NULL    Shelf Total             90
A       NULL        1       Shelf/Bin Total         49
A       NULL        2       Shelf/Bin Total         41
A       3           NULL    Shelf/Location Total    90
A       3           1       Regular Row             49
A       3           2       Regular Row             41

(12行受影响)
```

54

解析

GROUPING_ID接受几个列，并返回二进制列列表计算的整数值（我会逐步讲解它）。

这个查询先列出三个要在结果集中返回的非聚合列：

```
SELECT
   i.Shelf,
   i.LocationID,
   i.Bin,
```

然后，定义了评估GROUPING_ID返回值的CASE语句，GROUPING_ID以三列为列表：

```
CASE GROUPING_ID(i.Shelf,i.LocationID, i.Bin)
```

为了举例说明二进制到整数转换的概念，我们将关注一行，该行显示了由CUBE自动生成的货架A的全部总计：

```
Shelf   LocationID   Bin    Total
A       NULL         NULL   90
```

现在想象它下面还有一行，该行基于列是否为分组列显示了激活或禁用的比特值。GROUPING_ID透视出的Location和Bin都为比特值1，因为它们都不是这个特定行的分组列。对应这一行，Shelf是分组列：

```
Shelf   LocationID   Bin
A       NULL         NULL
0       1            1
```

将二进制011转换为整数，我将增加另一行，显示翻转位之下的整数值：

```
Shelf   LocationID   Bin
A       NULL         NULL
0       1            1
4       2            1
```

由于只有地区和箱柜有激活比特值，1加2的和为3，这是通过GROUPING_ID返回的这一行的值。因此各种分组的组合从二进制计算为整数。在随后的CASE语句中，3解释为货架的总和。

由于存在三列，在下面的WHEN/THEN中表现了各种潜在的聚合结果：

```
CASE GROUPING_ID(i.Shelf,i.LocationID, i.Bin)
   WHEN 1 THEN 'Shelf/Location Total'
   WHEN 2 THEN 'Shelf/Bin Total'
   WHEN 3 THEN 'Shelf Total'
   WHEN 4 THEN 'Location/Bin Total'
   WHEN 5 THEN 'Location Total'
   WHEN 6 THEN 'Bin Total'
   WHEN 7 THEN 'Grand Total'
 ELSE 'Regular Row'
```

55

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
END,
```

每个潜在的聚合结果的组合都控制在CASE语句中。剩下的查询包括对数量使用聚合函数，然后使用CUBE来查找货架、地区和箱柜的各种聚合组合：

```
   SUM(i.Quantity) Total
FROM   Production.ProductInventory i
WHERE i.LocationID IN (3) AND
      i.Bin IN (1,2)
GROUP BY CUBE (i.Shelf,i.LocationID, i.Bin)
ORDER BYi.Shelf, i.LocationID, i.Bin
```

1.12 公共表表达式

公共表表达式（CTE）和视图或衍生查询相似，允许创建一个临时的查询，该查询能在SELECT、INSERT、UPDATE或DELETE查询范围中引用。和衍生查询不同，不需要在每次使用CTE的时候复制查询定义多次。同时也能在CTE定义中使用局部变量——这是在视图定义中所不能做到的。

CTE的基本语法如下：

```
WITH expression_name [ ( column_name [,...n ] ) ]
AS ( CTE_query_definition )
```

表1-4描述了CTE的参数。

表1-4 CTE参数

参　数	描　述
expression_name	公共表表达式的名字
column_name [,...n]	表达式的唯一列名
CTE_query_definition	定义公共表表达式的SELECT查询

非递归的CTE在查询中不引用自己。它就像查询的临时结果集。递归的CTE的定义和非递归的CTE差不多，只是递归的CTE返回层次化的以及和自身相关的数据。和其他方式相比，使用CTE来表示递归数据能尽量减少需要的代码量。

接下来的两个技巧会演示非递归的和递归的CTE。

1.12.1 使用非递归的公共表表达式

这个公共表表达式的示例演示返回Purchasing.Vendor表中的供应商——返回根据名字排序的前5个和最后5个结果：

56

```
WITH VendorSearch (RowNumber, VendorName, AccountNumber)
AS
(
SELECT ROW_NUMBER() OVER (ORDER BY Name) RowNum,
       Name,
       AccountNumber
FROM Purchasing.Vendor
)

SELECT   RowNumber,
       VendorName,
       AccountNumber
FROM VendorSearch
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
WHERE RowNumber BETWEEN 1 AND 5
UNION
SELECT   RowNumber,
      VendorName,
      AccountNumber
FROM VendorSearch
WHERE RowNumber BETWEEN 100 AND 104
```

这个查询返回：

```
RowNumber   VendorName                  AccountNumber
1           A. Datum Corporation        ADATUM0001
2           Advanced Bicycles           ADVANCED0001
3           Allenson Cycles             ALLENSON0001
4           American Bicycles and Wheels AMERICAN0001
5           American Bikes              AMERICAN0002
100         Vista Road Bikes            VISTARO0001
101         West Junction Cycles        WESTJUN0001
102         WestAmerica Bicycle Co.     WESTAMER0001
103         Wide World Importers        WIDEWOR0001
104         Wood Fitness                WOODFIT0001

(10行受影响)
```

前面的示例使用了UNION，但是非递归的CTE也能像其他SELECT查询那样使用：

```
WITH VendorSearch (VendorID, VendorName)
AS
(
SELECT BusinessEntityID,
      Name
FROM Purchasing.Vendor
)

SELECT   v.VendorID,
      v.VendorName,
      p.ProductID,
      p.StandardPrice
FROM VendorSearch v
INNER JOIN Purchasing.ProductVendor p ON
   v.VendorID = p.VendorID
ORDER BY v.VendorName
```

57

这个查询返回如下的（部分）结果：

```
VendorID   VendorName          ProductID   StandardPrice
32         Advanced Bicycles   359         45.41
32         Advanced Bicycles   360         43.41
32         Advanced Bicycles   361         47.48
32         Advanced Bicycles   362         43.41
32         Advanced Bicycles   363         41.41
...

(460行受影响)
```

解析

在该技巧的第一个示例中，WITH定义了CTE的名字和它要返回的列名。这是一个非递归的CTE，因为CTE的数据不会再和自己联结。本例中的CTE只是使用了一个UNION两个数据集的查询：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
WITH VendorSearch (RowNumber, VendorName, AccountNumber)
```

在CTE中定义的列名可以和列在查询中实际的名字匹配——或者可以创建自己的别名。对于本例，Purchasing.Vendor的列名Name在CTE中被引用为VendorName。

然后，AS标记了CTE查询定义的开始：

```
AS
(
```

在圆括号中，查询使用了函数来返回结果集中有序的行号——根据供应商名字排序（有关ROW_NUMBER的更多信息参阅第8章）：

```
SELECT ROW_NUMBER() OVER (ORDER BY Name) RowNum,
      Name,
      AccountNumber
FROM Purchasing.Vendor
)
```

Purchasing.Vendor表的Name和AccountNumber也同样返回了。在关闭圆括号后CTE的定义就结束了。

在CTE定义之后就是使用CTE的查询。记住，如果SELECT、INSERT、UPDATE或DELETE语句需要引用一些或者全部CTE列，这些语句就必须放在CTE的定义之后：

```
SELECT   RowNumber,
   VendorName,
   AccountNumber
FROM VendorSearch
WHERE RowNumber BETWEEN 1 AND 5
```

SELECT中用的列名来自新的VendorSearch CTE。在WHERE子句中，第一个查询返回1到5行。接下来在第二个查询之前使用了UNION运算符：

```
UNION
```

第二个查询显示了开始5行和最后5行。VendorSearch CTE被引用了两次，但完整的查询定义只需要定义一次（和衍生查询不同），从而可以减少代码。

58

在技巧的第二个示例中，定义了一个简单的CTE，没有使用任何函数，只是Purchasing.Vendor表中的BusinessEntityID和Name：

```
WITH VendorSearch (VendorID, VendorName)
AS
(
SELECT BusinessEntityID,
      Name
FROM Purchasing.Vendor
)
```

在CTE定义之后的查询中，CTE VendorSearch像普通表一样进行联结（只是不用指定拥有者的架构）：

```
SELECT   v.VendorID,
      v.VendorName,
      p.ProductID,
      p.StandardPrice
FROM VendorSearch v
INNER JOIN Purchasing.ProductVendor p ON
   v.VendorID = p.BusinessEntityID
ORDER BY v.VendorName
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

警告 如果CTE是一批语句的一部分，那么CTE定义之前的语句必须以分号结尾。

注解 可以使用分号作为SQL Server语句的终止符。多数情况下这样做不是必需的，但是符合ANSI标准，你可以在一些微软的文档中看到这样的使用。

1.12.2 使用递归的公共表表达式

在这个示例中，新的Company表定义了一个假想的巨大的联合企业。每一个公司都有一个CompanyID和可选的ParentCompanyID。这个示例会演示如何使用递归CTE在结果中显示公司的层次。首先，创建表：

```
CREATE TABLE dbo.Company
   (CompanyID int NOT NULL PRIMARY KEY,
    ParentCompanyID int NULL,
    CompanyName varchar(25) NOT NULL)
```

接着，向新表中插入行（使用SQL Server 2008新语法的方式将在第2章中演示）：

```
INSERT dbo.Company (CompanyID, ParentCompanyID, CompanyName)
VALUES
   (1, NULL, 'Mega-Corp'),
   (2, 1, 'Mediamus-Corp'),
   (3, 1, 'KindaBigus-Corp'),
   (4, 3, 'GettinSmaller-Corp'),
   (5, 4, 'Smallest-Corp'),
   (6, 5, 'Puny-Corp'),
   (7, 5, 'Small2-Corp')
```

59

现在就是实际的例子：

```
WITH CompanyTree(ParentCompanyID, CompanyID, CompanyName, CompanyLevel)
AS
(
    SELECT    ParentCompanyID,
      CompanyID,
      CompanyName,
      0 AS CompanyLevel
    FROM dbo.Company
    WHERE ParentCompanyID IS NULL
    UNION ALL
    SELECT c.ParentCompanyID,
      c.CompanyID,
      c.CompanyName,
      p.CompanyLevel + 1
    FROM dbo.Company c
        INNER JOIN CompanyTree p
        ON c.ParentCompanyID = p.CompanyID
)
SELECT ParentCompanyID, CompanyID, CompanyName, CompanyLevel
FROM CompanyTree
```

这个查询返回：

```
ParentCompanyID   CompanyID   CompanyName          CompanyLevel
NULL              1           Mega-Corp            0
1                 2           Mediamus-Corp        1
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
1           3           KindaBigus-Corp     1
3           4           GettinSmaller-Corp  2
4           5           Smallest-Corp       3
5           6           Puny-Corp           4
5           7           Small2-Corp         4

(7行受影响)
```

解析

在这个示例中，首先定义了CTE名和列：

```
WITH CompanyTree(ParentCompanyID, CompanyID, CompanyName, CompanyLevel)
```

CTE查询定义由AS和一个左括号开始：

```
AS
(
```

SELECT子句由锚（**anchor**）SELECT语句开始。锚定义必须先定义。如果使用递归CTE，“锚”表示它定义了递归的根——在这里就是企业层次的最上面一级（没有父节点的Mega-Corp）。这个SELECT也包含了一个CompanyLevel列别名，并以数字0开始。这列用于在递归中显示某个公司在公司层次中的级别：

```
SELECT ParentCompanyID,
  CompanyID,
  CompanyName,
  0 AS CompanyLevel
FROM dbo.Company
WHERE ParentCompanyID IS NULL
```

接下来就是UNION ALL，把第二个递归查询联结到锚成员（UNION ALL而不是只有UNION，在递归CTE中，UNION ALL对于最后的锚成员和最前面的递归成员是必需的）：

```
UNION ALL
```

在这之后就是递归查询。和锚差不多，SELECT子句引用了dbo.Company表的ParentCompanyID、CompanyID和CompanyName。和锚不同，CTE列引用了p.CompanyLevel（从锚查询中），对层次的每一个级别加上1：

```
SELECT    c.ParentCompanyID,
  c.CompanyID,
  c.CompanyName,
  p.CompanyLevel+1
```

dbo.Company表被联结到CompanyTree CTE，联结CTE的递归查询的ParentCompanyID到CTE的CompanyID：

```
    FROM dbo.Company c
        INNER JOIN CompanyTree p
        ON c.ParentCompanyID = p.CompanyID
)
```

在关闭了CTE的定义后，查询从CTE中选择定义的列：

```
SELECT ParentCompanyID, CompanyID, CompanyName, CompanyLevel
FROM CompanyTree
```

在结果中，对于公司层次中的每一个级别，CTE使CompanyLevel列增加了。

然而，使用这种有用的新特性需要小心。如果你没有正确地创建递归CTE，可能会产生无限循环。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在测试的时候，要避免无限循环，可以使用本章前面提到的MAXRECURSION提示。

例如，如果希望阻止前面的示例超过2级，只需要在查询最后添加具有MAXRECURSION的OPTION子句：

```
WITH CompanyTree(ParentCompanyID, CompanyID, CompanyName, CompanyLevel) AS
(
    SELECT ParentCompanyID, CompanyID, CompanyName, 0 AS CompanyLevel
    FROM dbo.Company
    WHERE ParentCompanyID IS NULL
    UNION ALL
    SELECT c.ParentCompanyID, c.CompanyID, c.CompanyName, p.CompanyLevel + 1
    FROM dbo.Company c
        INNER JOIN CompanyTree p
        ON c.ParentCompanyID = p.CompanyID
)
SELECT ParentCompanyID, CompanyID, CompanyName, CompanyLevel
FROM CompanyTree
61 OPTION (MAXRECURSION 2)
```

这个查询返回：

```
ParentCompanyID   CompanyID   CompanyName          CompanyLevel
NULL              1           Mega-Corp            0
1                 2           Mediamus-Corp        1
1                 3           KindaBigus-Corp      1
3                 4           GettinSmaller-Corp   2
消息530，级别16，状态1，第2行
语句被终止。完成执行语句前已用完最大递归2。
```

作为最佳实践，根据对数据的理解设置MAXRECURSION。例如，如果你知道级别不能超过10级，就把MAXRECURSION设置成10。

提示 你也可以使用新的HierarchyID数据类型更轻松地遍历数据层次结构。想了解SQL Server 2008
62 更多的新数据类型，请参看第14章。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第2章 执行、捕获和跟踪数据修改

在本章中，我们会复习如何使用T-SQL的INSERT、UPDATE和DELETE命令来修改数据。在复习每个命令基础知识的同时也会涉及诸如调用存储过程插入数据、使用新的OPENROWSET BULK功能把图片文件导入表等具体技术。

本章会涉及的SQL Server 2008的新特性有如下几个。

- 利用一个INSERT语句插入多行，本章还会演示使用多行技术在SELECT子句中创建查询数据源（而不用创建永久表或临时表）。
- 新的赋值运算符，使用它可以以最少的代码量修改传入的数据值。
- 新的MERGE命令，使用它可以在一个代码块内合并并应用数据修改命令。
- 在维护SQL Server事务控制时，使用新的FILESTREAM属性向文件系统存储非结构化数据。
- 使用变更数据捕获（CDC）和更改跟踪内建功能来跟踪表数据变更的两个新选项。

在介绍新特性之前，我们会先复习INSERT的基本概念。

2.1 INSERT

简化的INSERT命令的语法如下：

```
INSERT
   [ INTO]
   table_or_view_name
   [ ( column_list ) ]
    VALUES (({DEFAULT | NULL | expression } [ ,...n ]) [ ,...n ])
```

表2-1描述了此命令的参数。

表2-1 INSERT命令参数

参　数	描　述
table_or_view_name	你要插入行的表或者可更新视图的名字
column_list	显式指定以逗号分隔的列名列表，目标表的这些列将被赋值
(DEFAULT \| NULL \| expression) [, ...n])	以逗号分隔的值列表，将被作为新的行插入表。在SQL Server 2008中，可在一条语句中插入多行。每一个值都可以是表达式、NULL值或者DEFAULT值（如果该列定义过默认值的话）

2.1.1 向表中插入一行

在这个技巧中，我将演示如何使用INSERT来向表（由table_name指定）中添加新行，这需要为

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

column_list指定数据应该插入到的那些列，并且在VALUES子句中设定一组相对应的以逗号分隔的待插入的值。比如，向AdventureWorks数据库的Production.Location表插入一行：

```
USE AdventureWorks
GO

INSERT Production.Location
(Name, CostRate, Availability)
VALUES ('Wheel Storage', 11.25, 80.00)
```

这个查询返回：

```
(1行受影响)
```

接下来通过一个查询搜索任何Name是Wheel Storage的行：

```
SELECT Name,
   CostRate,
   Availability
FROM Production.Location
WHERE Name = 'Wheel Storage'
```

这个查询返回：

```
Name            CostRate    Availability
Wheel Storage   11.25       80.00

(1行受影响)
```

解析

在这个技巧中，我们向Production.Location表插入了一个新行。

这个查询以INSERT命令开头，指定了接收插入数据的表名（INTO关键字是可选的）：

64

```
INSERT Production.Location
```

代码的下一行显式列出了目标表中待插入数据的那些列：

```
(Name, CostRate, Availability)
```

必须用逗号分隔每一列。列的次序无需是表中列的实际呈现次序，但VALUES子句中指定的值的次序必须和该列表中列的次序一一对应。如果INSERT语句按照表中列呈现的次序（或者内部物理次序）提供所有的值，那么列的列表可以不用写。然而对于你的产品代码，使用列的列表是必需的，特别是在表结构经常发生变动的情况下。那是因为，如果我们显式列出了这些列，为表添加新列就不用修改相关代码了（假设新的列有默认值）。

代码下一行是VALUES子句和一个要插入的以逗号分隔的值（表达式）列表：

```
VALUES ('Wheel Storage', 11.25, 80.00)
```

前面提到过，该列表中值的次序必须和列出的列的次序一一对应，或者如果没有列出列的话就和表中列的次序一样。

2.1.2　使用默认值插入行

在这个技巧中，我会介绍如何使用DEFAULT关键字向表中插入一行，从而为某个列（或者某些列）设定默认值。在前一个技巧中，已经向Production.Location表插入了一行。Production.Location表中还有两列，没有显式地在INSERT语句中引用。如果看看表2-2的列定义，你会发现LocationID和

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

ModifiedDate列没有包含在前面示例的INSERT中：

表2-2 Production.Location表定义

列 名	数据类型	可空性	默认值	标识列
LocationID	smallint	不允许		是
Name	dbo.Name（用户定义数据类型）	不允许		不是
CostRate	smallmoney	不允许	0.00	不是
Availability	decimal(8,2)	不允许	0.00	不是
ModifiedDate	datetime	不允许	GETDATE()（返回当前日期和时间的函数）	不是

注解 有关CREATE TABLE命令、IDENTITY列以及DEFAULT值的更多信息，请参阅第4章。

ModifiedDate列有一个默认值，对于新的行，如果列值没有显式插入，则使用当前的日期和时间填充。也可以使用INSERT来更新以这个列。例如：

```
INSERT Production.Location
(Name, CostRate, Availability, ModifiedDate)
VALUES ('Wheel Storage 2', 11.25, 80.00, '1/1/2005')
```

如果表中的某列已经指定了默认值，可以在VALUES子句中使用DEFAULT关键字来显式启用默认值。例如：

```
INSERT Production.Location
(Name, CostRate, Availability, ModifiedDate)
VALUES ('Wheel Storage 3', 11.25, 80.00, DEFAULT)
```

如果表中的每一列都设定了默认值，可以使用DEFAULT VALUES选项来插入一个完全使用默认值的行。例如：

```
INSERT dbo.ExampleTable
DEFAULT VALUES
```

解析

使用DEFAULT关键字可以在INSERT语句中为列显式设定默认值。DEFAULT VALUES关键字能用在INSERT语句中，用来显式设定所有列的默认值（假设表中的所有列都定义了默认值）。

然而，Production.Location表的LocationID列是IDENTITY列（不是默认列）。列的IDENTITY属性能让列自己填充一个自增的数字值。因为LocationID是IDENTITY列，数据库可以为这一列插入值所以INSERT语句不能显式添加其值（和默认值不同）。但是，在执行一些特殊的步骤后就可以显式为IDENTITY列插入值，在下个技巧中会详细介绍。

2.1.3 显式向一个 IDENTITY 列插入值

在这个技巧中，我会演示如何显式地向一个IDENTITY属性的列插入值。对于每一个向表中插入的行，一个拥有IDENTITY属性的列会根据一个数字种子值和增量值自动增加。IDENTITY列通常用作代理键（代理键是一个由数据库生成的唯一的主键，通常不包含任何业务意义，仅用于在表中确保唯一性）。

对于数据加载或者恢复的应用，可能需要为IDENTITY列手动插入显式值。例如，一个键值为4的行

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

意外删除了，而你需要手动重建那一行原始的业务信息并且保留4这个原始值，这样你就需要显式地把这个值插入到表中。

要显式地把一个数字值插入到一个具有IDENTITY属性的列中，必须使用SET IDENTITY_INSERT命令。语法如下：

```
SET IDENTITY_INSERT [ database_name . [ schema_name ] . ] table { ON | OFF }
```

表2-3描述了这个命令的参数。

表2-3　SET IDENTITY_INSERT命令

参　数	描　述
[database_name.[schema_name] .] table	可选的数据库名、可选的架构名和必需的表名。在这个表上将允许对IDENTITY属性的列显式插入一个值
ON \| OFF	当设置成ON的时候，允许插入显式值。当设置成OFF的时候，不允许插入显式值

这个技巧将演示如何向表中的IDENTITY列显式插入一个值。下面的查询首先演示了如果不使用IDENTITY_INSERT，向IDENTITY列显式插入一个值会发生什么情况：

```
INSERT HumanResources.Department
(DepartmentID, Name, GroupName)
VALUES (17, 'Database Services', 'Information Technology')
```

这个查询返回一个错误，不允许显式地为标识列插入一个值：

```
消息544，级别16，状态1，第2行
当IDENTITY_INSERT设置为OFF时，不能为表'Department'中的标识列插入显式值。
```

使用SET IDENTITY_INSERT可以去除这个限制，如下例所示：

```
SET IDENTITY_INSERT HumanResources.Department ON

INSERT HumanResources.Department
(DepartmentID, Name, GroupName)
VALUES (17, 'Database Services', 'Information Technology')

SET IDENTITY_INSERT HumanResources.Department OFF
```

解析

在这个技巧中，我们在插入之前把这个属性设置为ON：

```
SET IDENTITY_INSERT HumanResources.Department ON
```

然后使用17这个值执行INSERT。当向一个标识列进行插入操作的时候，也需要在INSERT table_name子句后显式列出列名。

```
INSERT HumanResources.D epartment
(DepartmentID, Name, GroupName)
VALUES (17, 'Database Services', 'Information Technology')
```

由于插入的值大于当前的标识值，表的新的插入会自动使用新值作为标识种子。

一旦完成显式插入后，应该把IDENTITY_INSERT设为OFF：

```
SET IDENTITY_INSERT HumanResources.Department OFF
```

因为在一个会话（数据库连接会话）中同一时间只能有一个表IDENTITY_INSERT ON。关闭会话将会移除ON属性，将其设置回OFF。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

2.1.4 在表中插入拥有 uniqueidentifier 列的行

在这个技巧中，会介绍如何向表中插入使用uniqueidentifier列的行。当你必须在几个SQL Server实例中保证标识符唯一时，这个数据类型是有用的。例如，如果你将10个远程SQL Server实例生成的记录合并到一个SQL Server实例中，使用IDENTITY值会有主键冲突的风险。而使用uniqueidentifier可以避免这个风险。

uniqueidentifier数据类型存储了一个16字节的全球唯一标识符（GUID），它经常用来确保同一数据库甚至不同数据库跨表的唯一性。GUID通常可以作为整数值键的替补，但是对于大的表，相对于整数值来说，它们的宽度有时候会导致较低的查询性能。

要为新的插入生成这个值可以使用NEWID系统函数。NEWID生成了一个唯一的uniqueidentifier数据类型的值，如下面的技巧所示：

```
INSERT Purchasing.ShipMethod
(Name, ShipBase,  ShipRate, rowguid)
VALUES('MIDDLETON CARGO TS1',  8.99, 1.22,  NEWID())

SELECT rowguid,  name
FROM Purchasing.ShipMethod
WHERE Name = 'MIDDLETON CARGO TS1'
```

这个查询返回：

```
Rowguid                                  name
174BE850-FDEA-4E64-8D17-C019521C6C07     MIDDLETON CARGO TS1
```

解析

Purchasing.ShipMethod表的rowguid列是uniqueidentifier数据类型的列。下面是表定义的一部分：

```
rowguid uniqueidentifier ROWGUIDCOL NOT NULL DEFAULT (newid()),
```

要为新插入的行生成一个新的uniqueidentifier数据类型的值，可以在VALUES子句中使用NEWID()函数：

```
VALUES('MIDDLETON CARGO TS1',  8.9 9,  1.2 2,  NEWID())
```

查询刚创建的行，rowguid是一个uniqueidentifier的值174BE850-FDEA-4E64-8D17-C019521C6C07（如果你自己实践的话会得到另外一个值，因为NEWID每次执行的时候都会产生一个新值）。

2.1.5 使用 INSERT...SELECT 语句插入行

前面的技巧演示了如何插入一行数据。在这个技巧中，将介绍如何使用INSERT...SELECT来向表中插入多行。实现INSERT...SELECT操作的语法如下：

```
INSERT
   [ INTO]
   table_or_view_name
   [ ( column_list ) ]
    SELECT column_list FROM data_source
```

使用INSERT...SELECT的语法基本和插入单行的语法基本一样，只是没有VALUES子句，而是指定一个SELECT查询把它返回的列和行填充到表或者可更新的视图中。SELECT查询能基于一个或多个数据源，只要列表的数据类型和目标表的数据类型一致就可以了。

就本例而言，我们将创建一个新的表来存放这些行。这个示例把HumanResources.Shift表中的值

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

68 填充到新的dbo.Shift_Archive表中：

```
CREATE TABLE [dbo]. [Shift_Archive](
  [ShiftID] [tinyint] NOT NULL,
  [Name] [dbo]. [Name] NOT NULL,
  [StartTime] [datetime] NOT NULL,
  [EndTime] [datetime] NOT NULL,
  [ModifiedDate] [datetime] NOT NULL DEFAULT (getdate()),
 CONSTRAINT [PK_Shift_ShiftID] PRIMARY KEY CLUSTERED
([ShiftID] ASC)
)
GO
```

下面，执行INSERT...SELECT：

```
INSERT dbo.Shift_Archive
(ShiftID, Name, StartTime, EndTime, ModifiedDate)
SELECT ShiftID, Name, StartTime, EndTime, ModifiedDate
FROM HumanResources.Shift
ORDER BY ShiftID
```

结果显示插入了3行：

```
(3行受影响)
```

接下来执行一个查询来确认Shift_Archive表中插入的行：

```
SELECT ShiftID, Name
FROM Shift_Archive
```

这个查询返回：

```
ShiftID     Name
1           Day
2           Evening
3           Night

(3行受影响)
```

解析

使用INSERT...SELECT语句，可以根据一个SELECT查询向表中插入多行。和常规的单值INSERT一样，先是使用INSERT table_name和要插入到表中的列（圆括号中的部分）：

```
INSERT Shift_Archive
(ShiftID, Name, StartTime, EndTime, ModifiedDate)
```

接着就是用于填充表的查询。SELECT语句返回的列必须和新表中列的次序一样。同样，各列的数据类型也必须和相关的INSERT列兼容：

```
SELECT ShiftID, Name, StartTime, EndTime, ModifiedDate
FROM HumanResources.Shift
ORDER BY ShiftID
```

69 由于没有指定列列表，因此SELECT语句必须为要插入数据的表的所有列提供数据。

2.1.6 调用存储过程插入数据

在这个技巧中，会演示如何通过使用存储过程插入表数据。存储过程把一个或多个T-SQL语句放在一个逻辑单元中，并将其作为一个对象存储在SQL Server数据库中。存储过程允许创建更灵活的结

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

果集（例如可以在返回最终结果集之前在临时表中创建几个过渡的结果集）。返回结果集的报表系统的存储过程同样能用于INSERT...EXEC，如果希望在表中保留SQL Server信息的话，这会很有用。

这个技巧也会教你如何把存储过程的输出添加到表中。只有当存储过程中定义的SELECT命令返回数据，并且结果集（甚至是多个结果集）与必须提供给INSERT的值的个数相匹配的时候，存储过程才能这么用。

注解 *有关存储过程的更多信息请参阅第10章。*

从存储过程插入数据的语法如下：

```
INSERT
   [ INTO]
   table_or_view_name
   [ ( column_list ) ]
    EXEC stored_procedure_name
```

这个语法和前面演示的INSERT示例几乎一样，只不过这次是通过执行一个存储过程来获取数据。

在本例中，我们创建了一个存储过程，根据传入存储过程的开始日期和结束日期从Production.TransactionHistory表中返回一些行。存储过程返回的这些结果同样是Production.TransactionHistoryArchive表中不存在的那些行：

```
CREATE PROCEDURE dbo.usp_SEL_Production_TransactionHistory
   @ModifiedStartDT datetime,
   @ModifiedEndDT datetime
AS

SELECT TransactionID, ProductID, ReferenceOrderID, ReferenceOrderLineID,
 TransactionDate, TransactionType, Quantity, ActualCost, ModifiedDate
FROM Production.TransactionHistory
WHERE ModifiedDate BETWEEN @ModifiedStartDT AND @ModifiedEndDT AND
    TransactionID NOT IN
      (SELECT TransactionID
         FROM Production.TransactionHistoryArchive)

GO
```

接着，使用一个示例测试存储过程，提前检查哪些行会被插入：

```
EXEC dbo.usp_SEL_Production_TransactionHistory '6/2/04', '6/3/04'
```

根据传入存储过程的日期范围，这个查询返回568行。在下面的示例中，存储过程用来把568行插入到表Production.TransactionHistoryArchive中：


```
INSERT Production.TransactionHistoryArchive
(TransactionID, ProductID, ReferenceOrderID, ReferenceOrderLineID, TransactionDate,
TransactionType, Quantity, ActualCost, ModifiedDate)
EXEC dbo.usp_SEL_Production_TransactionHistory '6/2/04', '6/3/04'
```

解析

这个示例演示了使用存储过程以及INSERT和EXEC向表填充数据。首先是INSERT和要插入数据的表名：

```
INSERT Production.TransactionHistoryArchive
```

接着就是要插入的列：

```
(TransactionID, ProductID, ReferenceOrderID, ReferenceOrderLineID,
TransactionDate, TransactionType, Quantity, ActualCost, ModifiedDate)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

最后就是用EXEC语句来执行存储过程。存储过程名后面跟随的是存储过程需要的参数：

```
EXEC usp_SEL_Production_TransactionHistory '6/2/04', '6/3/04'
```

2.1.7　通过 VALUES 插入多个行

SQL Server 2008引入了一个新功能：可以使用单个INSERT命令插入多行，而不用使用子查询或调用存储过程。这可以为应用程序减少必需的代码量，也可以减少执行命令的数量。基本方法是使用VALUES来分组，然后指定一个或更多行及相应的列值，如下面的技巧所演示的：

```
-- 创建一个查找表
CREATE TABLE HumanResources.Degree
   (DegreeID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
    DegreeNM varchar(30) NOT NULL,
    DegreeCD varchar(5) NOT NULL,
    ModifiedDate datetime NOT NULL)
GO

INSERT HumanResources.Degree
(DegreeNM, DegreeCD, ModifiedDate)
VALUES
 ('Bachelor of Arts', 'B.A.', GETDATE()),
 ('Bachelor of Science', 'B.S.', GETDATE()),
 ('Master of Arts', 'M.A.', GETDATE()),
 ('Master of Science', 'M.S.', GETDATE()),
 ('Associate''s Degree', 'A.A.', GETDATE())
GO
```

查询返回如下的查询输出：

71

```
(5行受影响)
```

解析

在这个技巧中，演示了从一个INSERT语句插入多个行。首先创建一个保存大学学位类型信息的表。然后以常规的方式使用INSERT，显示要为每行传入值的列名称：

```
INSERT HumanResources.Degree
(DegreeNM, DegreeCD, ModifiedDate)
```

然后，在VALUES子句中为每个学位类型指定新的行。每行都有3列，并且这些都被括在圆括号中：

```
VALUES
 ('Bachelor of Arts', 'B.A.', GETDATE()),
 ('Bachelor of Science', 'B.S.', GETDATE()),
 ('Master of Arts', 'M.A.', GETDATE()),
 ('Master of Science', 'M.S.', GETDATE()),
 ('Associate''s Degree', 'A.A.', GETDATE())
GO
```

使用这个新特性，不用重复输入初始的INSERT表名和列的列表就可以插入多行。在包含一个数据库架构和一组相关查找值的自定义应用程序中，这可能会是有用的。比起在初始化脚本中手写50条INSERT语句，创建一个指定多行的INSERT会好得多。这也可以避免你从外部源导入行集。

2.1.8　将 VALUES 作为表源来使用

前面的技巧演示了如何不必重复输入初始的INSERT表名和列的列表就插入多行。使用这个SQL Server 2008的新特性，你也可以在SELECT语句中的FROM子句中引用VALUES列表。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个技巧演示了如何不使用永久表或临时表来引用一个结果集。随后的查询演示了在5行的结果集中列出各种大学学位——不需要把行放入表中或从子查询中引用：

```
SELECT DegreeNM, DegreeCD, ModifiedDT
FROM
(VALUES
 ('Bachelor of Arts', 'B.A.', GETDATE()),
 ('Bachelor of Science', 'B.S.', GETDATE()),
 ('Master of Arts', 'M.A.', GETDATE()),
 ('Master of Science', 'M.S.', GETDATE()),
 ('Associate''s Degree', 'A.A.', GETDATE()))
 Degree (DegreeNM, DegreeCD, ModifiedDT)
```

查询返回：

```
DegreeNM              DegreeCD    ModifiedDT
Bachelor of Arts      B.A.        2007-08-21 19:10:34.667
Bachelor of Science   B.S.        2007-08-21 19:10:34.667
Master of Arts        M.A.        2007-08-21 19:10:34.667
Master of Science     M.S.        2007-08-21 19:10:34.667
Associate's Degree    A.A.        2007-08-21 19:10:34.667

(5行受影响)
```

解析

这个技巧演示了使用SQL Server 2008的新技术来返回结果集在存储设备中保存行。分解这个查询，SELECT子句的第一行列出了列的名称：

```
SELECT DegreeNM, DegreeCD, ModifiedDT
```

在引用的表中没有实际的列名——而是后面在查询本身定义的别名。

下一行定义了数据源的FROM子句，随后是圆括号括起来的VALUES关键字：

```
FROM
(VALUES
```

后面的几行代码列出了希望这个查询返回的行（与前面的技巧中如何在一条INSERT语句中插入多个行相似）：

```
 ('Bachelor of Arts', 'B.A.', GETDATE()),
 ('Bachelor of Science', 'B.S.', GETDATE()),
 ('Master of Arts', 'M.A.', GETDATE()),
 ('Master of Science', 'M.S.', GETDATE()),
 ('Associate''s Degree', 'A.A.', GETDATE())
)
```

最后，在行的列表结尾处的关闭圆括号后面，为数据源和要返回的每个列的相关列名定义了名称（为了在SELECT子句中引用）：

```
Degree (DegreeNM, DegreeCD, ModifiedDT)
```

使用这个新技术，不必实际创建临时或永久表就能指定表源的行。

2.2 UPDATE

UPDATE语句的基本语法如下：

```
UPDATE <table_or_view_name>
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SET column_name = {expression | DEFAULT | NULL} [ ,...n ]
WHERE <search_condition>
```

表2-4描述了此命令的参数。

表2-4　UPDATE命令参数

参　数	描　述
table_or_view_name	包含需要更新的数据的表或者可更新视图的名字
column_name ={expression \| DEFAULT \| NULL}	要更新的一列或者多列的名字。能设置为一个expression、列的DEFAULT值或者NULL
search_condition	定义了需要修改的行的搜索条件。如果不指定，则表或可更新视图的所有行都会被修改

2.2.1　更新一行

在这个技巧中，我会演示如何使用UPDATE语句来修改数据。使用UPDATE语句，你能对一列或多列、一行或多行进行修改。

在这个示例中，我们通过指定SpecialOfferID来更新一行，SpecialOfferID是表的主键（更多有关主键的信息请参阅第4章）。在实施这个更新之前，我们先查询计划修改的行：

```
SELECT DiscountPct
FROM Sales.SpecialOffer
WHERE SpecialOfferID = 10
```

查询返回：

```
DiscountPct
0.50
```

现在我实施修改操作：

```
UPDATE Sales.SpecialOffer
SET DiscountPct = 0.15
WHERE SpecialOfferID = 10
```

更新操作后我们对特定行进行查询，以确认DiscountPct的值确实修改了：

```
SELECT DiscountPct
FROM Sales.SpecialOffer
WHERE SpecialOfferID = 10
```

这个查询返回：

```
DiscountPct
0.15
```

解析

在这个示例中，查询从UPDATE和表名Sales.SpecialOffer开始：

```
UPDATE Sales.SpecialOffer
```

然后，使用SET，后面跟随需要修改的列，然后是一个等号运算符把DiscountPct设置成0.15这个值。对照这个技巧开始的语法，此示例是将列设置为一个表达式的值而不是DEFAULT或者NULL值。

```
SET DiscountPct = 0.15
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

因为UPDATE子句在表级别而不是行级别进行操作，如果就这样结束这个查询的话，Sales.SpecialOffer表中所有的行都会被修改。但是我们这个查询的目的是仅更新某个产品的折扣百分率，这需要使用WHERE子句来实现：

```
WHERE SpecialOfferID = 10
```

在执行查询后，只有一行被修改。如果有多行匹配WHERE子句中的搜索条件，那些行同样也会被修改。

提示 在UPDATE之前执行一个和UPDATE具有同样的FROM和WHERE子句的SELECT查询能让你知道哪些行会被更新（这个额外验证可以确保你将要更新的是正确的行）。万一你的修改是意外的，这时候就可以使用事务进行回滚。关于事务的更多信息请参阅第3章。

2.2.2 根据 FROM 和 WHERE 子句更新行

在这个技巧中，将介绍如何基于FROM子句和相关的WHERE子句搜索条件使用UPDATE语句更新行。基本语法比前面一个示例稍复杂，如下所示：

```
UPDATE <table_or_view_name>
SET column_name = {expression | DEFAULT | NULL} [ ,...n ]
FROM <table_source>
WHERE <search_condition>
```

FROM和WHERE子句不是必需的。然而，如果要通过联结一个或者多个表来确切地限定哪些行被更新，通常会使用它们。

在这个示例中，假设Production.Product表中有一个产品“Full-Finger Gloves, M”，这个产品每一位顾客最多只允许购买两个。为了达到这个查询的目的，任何购物车中只要这个产品的数量超过两个，都需要立即调整为所要求的限制：

```
UPDATE Sales.ShoppingCartItem
SET   Quantity =2,
  ModifiedDate = GETDATE()
FROM Sales.ShoppingCartItem c
INNER JOIN Production.Product p ON
  c.ProductID = p.ProductID
WHERE p.Name = 'Full-Finger Gloves, M ' AND
  c.Quantity > 2
```

解析

一点一点来分析这段代码，第一行显示了需要更新的表：

```
UPDATE Sales.ShoppingCartItem
```

接着，在SET子句中指定需要更新的列：

```
SET   Quantity =2,
  ModifiedDate = GETDATE()
```

然后，我们在可选的FROM子句中通过ProductID来联结Sales.ShoppingCartItem和Production.Product两个表。可以看到，待更新的对象同样能被FROM子句引用。UPDATE和FROM引用了同样的表：

```
FROM Sales.ShoppingCartItem c
INNER JOIN Production.Product p ON
  c.ProductID = p.P roductID
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

75 在FROM子句中使用更新表的目的在于联结到其他表。可想而知，其他那些联结的表会用来过滤更新的行或者为更新行提供值。如果在FROM子句中的自联结不止一次引用到要更新的表，那么最少有一个对该对象的引用不可以指定表别名，而所有其他对象引用必须使用别名。

WHERE子句指出了，Sales.ShoppingCartItem表中的“Full-Finger Gloves, M”产品只有在购物数量超过2个的时候才应该被修改：

```
WHERE p.Name = 'Full-Finger Gloves, M ' AND
  c.Quantity > 2
```

2.2.3　更新大值数据类型的列

在这个技巧中，将介绍如何修改大值数据类型列的值。SQL Server在其早期版本中就引入了新的大值数据类型，旨在代替不推荐的text、ntext和image数据类型。这些新的数据类型包括：

- varchar(max)，保存非Unicode的变长数据；
- nvarchar(max)，保存Unicode的变长数据；
- varbinary(max)，保存变长的二进制数据。

这些数据类型能存储高达2^31-1字节的数据（有关数据类型的更多信息，请参阅第4章）。

旧的text和image数据类型的一个主要缺点是，需要使用诸如WRITETEXT和UPDATETEXT等独立的函数来操作image/text数据。使用新的大值数据类型，就可以直接使用常规的INSERT和UPDATE操作。

插入大值数据类型的语法和普通的插入没有什么两样。但是，对于更新大值数据类型，UPDATE命令包含了一个.WRITE方法：

```
UPDATE <table_or_view_name>
SET column_name = .WRITE ( expression , @Offset , @Length )
FROM <table_source>
WHERE <search_condition>
```

表2-5描述了.WRITE方法的参数。

表2-5　使用.WRITE子句的UPDATE命令

参　数	描　述
expression	这个表达式定义了要存入列的文本块
@Offset	@Offset定义了新的文本在现有数据中存放的起始位置。如果@Offset是NULL，就代表新的表达式会追加到列的最后（同时会忽略第二个@Length参数）
@Length	@Length定义了需要覆盖部分的长度

这个示例从创建一个叫RecipeChapter的表开始：

```
CREATE TABLE dbo.RecipeChapter
  (ChapterID int NOT NULL,
   Chapter varchar(max) NOT NULL)
GO
```

76 然后，向表中插入一行。注意到，要插入到Chapter varchar(max)列的字符串没有什么特殊的：

```
INSERT dbo.RecipeChapter
(ChapterID, Chapter)
VALUES
(1, 'At the beginning of each chapter you will notice
that basic concepts are covered first.' )
```

下个示例就是更新刚插入的行，在现有的句子最后添加一个句子：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
UPDATE RecipeChapter
SET Chapter .WRITE (' In addition to the basics, this chapter will also provide
recipes that can be used in your day to day development and administration.' ,
NULL, NULL)
WHERE ChapterID = 1
```

接着，把那行中的短语“day to day”替换成一个单词“daily”：

```
UPDATE RecipeChapter
SET Chapter .WRITE('daily', 181, 10)
WHERE ChapterID = 1
```

最后，返回那行的结果：

```
SELECT Chapter
FROM RecipeChapter
WHERE ChapterID = 1
```

这个查询返回：

```
Chapter
At the beginning of each chapter you will notice that basic concepts
are covered first.
In addition to the basics, this chapter will also provide recipes
 that can be used in your daily development and administration.
```

解析

在这个技巧中，我们首先创建了一个表，用于存放书中每一章的描述信息。Chapter列使用了varchar(max)数据类型：

```
CREATE TABLE RecipeChapter
  (ChapterID int NOT NULL,
   Chapter varchar(max) NOT NULL)
```

然后，插入了一个新行。注意到，插入大值数据类型的语法和插入数据到一个普通的非大值数据类型没有什么两样：

```
 INSERT RecipeChapter
(ChapterID, Chapter)
VALUES
(1, 'At the beginning of each chapter you will
notice that basic concepts are covered first.' )
```

接着，对RecipeChapter表执行了UPDATE，在第一句话的最后添加第二个句子：

```
UPDATE RecipeChapter
```

在SET命令后跟随要更新的列名（Chapter）和新的.WRITE命令。.WRITE命令后跟一个左圆括号、一个单引号和需要追加在列最后的句子：

```
SET Chapter .WRITE(' In addition to the basics,
this chapter will also provide recipes that can be
used in your day to day development and administration.' ,
NULL, NULL)
```

WHERE子句指出了只有ChapterID为1的Chapter列才被修改：

```
WHERE ChapterID = 1
```

下一个.WRITE的示例演示了替换列正文中的数据。在示例中，表达式day to day被替换成daily。@Offset和@Length这两个bigint值对于varbinary(max)和varchar(max)数据类型来说是以字节为度量单

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

位。对于nvarchar(max)，这些参数度量了实际的字符数。例如，.WRITE有这样的@Offset值（文本的第178字节）和@Length（10字节长）值：

```
UPDATE RecipeChapter
SET Chapter .WRITE('daily', 181, 10)
WHERE ChapterID = 1
```

2.2.4　使用 OPENROWSET 和 BULK 插入或更新图片文件

这个技巧会演示如何把文件系统中的图片文件插入或更新到SQL Server表中。在SQL Server 早期的版本中，要想把图片添加到表中，通常需要使用外部的应用工具或者脚本，仅仅使用T-SQL没有什么优雅的方式来插入图形。

到了SQL Server 2005和SQL Server 2008，可以结合使用UPDATE和OPENROWSET来把图片导入表中。我们能使用OPENROWSET把文件导入到某一行的某一列。本技巧中会用到的OPENROWSET的基本语法如下：

```
OPENROWSET
( BULK 'data_file' ,
       SINGLE_BLOB | SINGLE_CLOB | SINGLE_NCLOB )
```

表2-6描述了这个命令的参数。

表2-6　OPENROWSET命令参数

参　数	描　述
data_file	指定要读取的文件的名字和路径
SINGLE_BLOB \| SINGLE_CLOB \| SINGLE_NCLOB	指定SINGLE_BLOB对象是导入到varbinary(max)数据类型中，指定SINGLE_CLOB是导入ASCII数据到varchar(max)数据类型中，指定SINGLE_NCLOB是导入到nvarchar(max)Unicode数据类型中

78　**注解**　OPENROWSET的语法细节请参阅第27章。

这个技巧的第一部分是创建一个用于存储BMP图片文件的新表：

```
CREATE TABLE dbo.StockBmps
  (StockBmpID int NOT NULL,
   bmp varbinary(max) NOT NULL)
GO
```

然后，向表中插入一个包含图片文件的行：

```
INSERT dbo.StockBmps
(StockBmpID,  bmp)
SELECT  1,
    BulkColumn
FROM OPENROWSET(BULK
'C:\Apress\StockPhotoOne.bmp', SINGLE_BLOB) AS x
```

接着使用一个查询查看表中的这行：

```
SELECT bmp
FROM StockBmps
WHERE StockBmpID = 1
```

这个查询返回如下的（部分）结果：

```
bmp
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
0x424D3656000000000000360400002800000007D000000A40000000100080000000000052000000000
0000000000000010000000100001B71900057575E00EFEFEF000F0B0C0023A7D30028D2FF001A5B7
1005473A1008C8C8C00B3B3B300208BB00031303100D1D1D1005896B20018425600112C3500777D
7B00474F9100A089660078CDDD0071AFC6009D9D9D0045444A00686B6F00728FAD0077998C001
C1D1E000904050008030400050100026C4FF
```

本技巧的最后一个示例更新一个现有的BMP文件，将其修改成另外一个BMP文件：

```
UPDATE dbo.StockBmps
SET bmp =
(SELECT BulkColumn
FROM OPENROWSET(BULK 'C:\Apress\StockPhotoTwo.bmp', SINGLE_BLOB) AS x)
WHERE StockBmpID =1
```

解析

在这个技巧中，演示了使用OPENROWSET和BULK选项将一个包含BMP图片文件的行插入到表中，以及把它更新成另外一个BMP文件的方法。

首先，我们创建了一个表来保存这个BMP文件，使用的是varbinary(max)数据类型：

```
CREATE TABLE dbo.StockBmps
  (StockBmpID int NOT NULL,
   bmp varbinary(max) NOT NULL)
```

接着，使用INSERT来插入一个新行：

```
INSERT dbo.StockBmps
(StockBmpID, bmp)
```

然后使用一个SELECT查询和OPENROWSET函数来获取文件数据，组成INSERT的内容。查询中用到的BulkColumn表示从OPENROWSET数据源获取的、待插入到varbinary(max)行中的varbinary值：

```
SELECT 1,
   BulkColumn
```

在FROM子句中调用了OPENROWSET。OPENROWSET允许我们从数据源中访问远程数据：

```
FROM OPENROWSET(BULK
'C:\Apress\StockPhotoOne.bmp', SINGLE_BLOB) AS x
```

在函数中使用了BULK选项，后面跟随文件名和SINGLE_BLOB关键字。OPENROWSET中的BULK选项表示数据会从文件读取（本例中在BULK后指定了BMP文件）。SINGLE_BLOB开关指示OPENROWSET以单行单列的varbinary(max)形式返回数据文件的内容。

这个技巧同样演示了从一个外部文件更新varbinary(max)列。UPDATE指定了StockBmps表并且使用SET来更新bmp列：

```
UPDATE StockBmps
SET bmp =
```

子查询中的表达式把图片从原先的StockphotoOne.bmp设置成新的StockphotoTwo.bmp。它几乎使用了和前面INSERT相同的语法，只不过这次从SELECT返回的值只有BulkColumn列：

```
(SELECT BulkColumn
FROM OPENROWSET(BULK 'C:\Apress\StockPhotoTwo.bmp', SINGLE_BLOB) AS x)
```

这样，机器上的图片文件就以varbinary(max)数据保存到了那行的列中。

2.2.5 在维护 SQL Server 事务控制时向文件系统存储非结构化数据

SQL Server 2008引入了FILESTREAM新属性，可以将它应用到varbinary(max)数据类型上。使用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

FILESTREAM存储值到文件系统时可以超过2GB的限制，还可以带来SQL Server的对文件关系处理的优点，虽然实际上文件还是存储在文件系统中的。BACKUP和RESTORE操作不仅能对数据库数据进行维护，也可以对保存在文件系统中的文件进行维护，因此可以控制存储结构化和非结构化数据的应用程序的端到端数据的可复原性。FILESTREAM把NT文件系统流的性能优点与SQL Server事务一致性能力完美地结合到了一起。

T-SQL用来定义用来FILESTREAM属性并可以用来处理数据，而在应用程序实施实际的读写操作时应该首选的是Win32流API（具体来说使用OpenSqlFilestream API）。尽管演示Win32和可应用API的实现超出了本书范围，但我会在本技巧中一步步介绍如何设置具有FILESTREAM属性的数据和表，如何插入新行，以及使用查询来获得调用OpenSqlFilestream API时必要的路径和事务标记信息。

提示 必须在Windows和SQL Server范围内都配置FILESTREAM。要在Windows范围内启用FILESTREAM并且定义相关的文件共享，需要使用SQL Server配置管理器。而要在SQL Server实例级别启用FILESTREAM，需要使用sp_configure和filestream_access_level选项。

为了确认SQL Server 实例中是否配置了FILESTREAM，可以使用SERVERPROPERTY函数和3个不同属性
80 来验证这个设置，这3个属性分别是文件流共享的文件共享名称、相关效果值和实际配置的值：

```
SELECT SERVERPROPERTY('FilestreamShareName') ShareName,
       SERVERPROPERTY('FilestreamEffectiveLevel') EffectiveLevel,
       SERVERPROPERTY('FilestreamConfiguredLevel') ConfiguredLevel
```

这个查询返回：

```
ShareName     EffectiveLevel     ConfiguredLevel
AUGUSTUS      3                  3
```

然后，将创建一个新数据库，它有一个包含FILESTREAM数据的文件组。

提示 更多关于CREATE DATABASE命令的内容请参阅第22章。

与CREATE DATABASE中常规的文件/文件组赋值不同，我为文件组指定路径，然后指定SQL Server要在文件系统上创建的文件夹的名称，并包含所有与数据库相关的FILESTREAM文件：

```
USE master
GO

CREATE DATABASE PhotoRepository ON PRIMARY
( NAME = N'PhotoRepository',
  FILENAME = N'C:\Apress\MDF\PhotoRepository.mdf' ,
  SIZE = 3048KB ,
  FILEGROWTH = 1024KB ),
  FILEGROUP FS_PhotoRepository CONTAINS FILESTREAM
  (NAME = 'FG_PhotoRepository',
   FILENAME = N'C:\Apress\FILESTREAM')
 LOG ON
( NAME = N'PhotoRepository_log',
  FILENAME = N'C:\Apress\LDF\PhotoRepository_log.ldf' ,
  SIZE = 1024KB ,
  FILEGROWTH = 10%)
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下面创建一个将用来存储书籍封面图片的新表。将BookPhotoFile列指定为varbinary(max)数据类型，随后是FILESTREAM属性：

```
USE PhotoRepository
GO

CREATE TABLE dbo.BookPhoto
   (BookPhotoID uniqueidentifier ROWGUIDCOL NOT NULL PRIMARY KEY,
    BookPhotoNM varchar(50) NOT NULL,
    BookPhotoFile varbinary(max) FILESTREAM)
GO
```

现在表已经创建好了，下面将使用常规的INSERT命令插入一个新行，并且使用OPENROWSET（在前面的技巧中有演示）来导入文件：

```
INSERT dbo.BookPhoto
(BookPhotoID, BookPhotoNM, BookPhotoFile)
SELECT NEWID(),
       'SQL Server 2008 Transact-SQL Recipes cover',
        BulkColumn
FROM OPENROWSET(BULK
'C:\Apress\TSQL2008Recipes.bmp', SINGLE_BLOB) AS x
```

如果查看C:\Apress\FILESTREAM目录下面，会看到新的子目录和新的文件。在这个例子中，我在我的服务器上的C:\Apress\FILESTREAM\33486315-2ca1-43ea-a50e-0f84ad8c3fa6\e2f310f3-cd21-4f29-acd1-a0a3ffb1a681路径下看到名称为00000012-000000e1-0002的新文件。使用FILESTREAM创建的文件只有在T-SQL的上下文和相应的Win32 API中才可以访问。

在插入值之后，运行SELECT来查看表中的内容：

```
SELECT BookPhotoID, BookPhotoNM, BookPhotoFile
FROM dbo.BookPhoto
```

这个查询返回：

BookPhotoID	BookPhotoNM	BookPhotoFile
236E5A69-53B3-4CB6-9F11-EF056082F542	SQL Server 2008 T-SQL Recipes cover	0x424D3656000000000000360400002800000007000000A40000000100080000000000005200000000000000000000000001000000010000276B8E0026B0ED005B5D6900EEEEEE00528CA2000E0A0B001C597900B3B3B3008B8A8D00D1D1D1002AC6FF002394C7002280AB002C2A2B00193F560066ADBD0025A4DC001128340005E

现在假设有一个使用OLEDB查询SQL Server实例的应用程序，为了使用应用程序来实现“流”，需要收集文件系统文件的适当信息。

先打开事务，并使用varbinary列的PathName()方法得到文件的逻辑路径名称：

```
BEGIN TRAN

SELECT BookPhotoFile.PathName()
FROM dbo.BookPhoto
WHERE BookPhotoNM = 'SQL Server 2008 Transact-SQL Recipes cover'
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
\\CAESAR\AUGUSTUS\v1\PhotoRepository\dbo\BookPhoto\BookPhotoFile\
236E5A69-53B3-4CB6-9F11-EF056082F542
```

然后，获得事务的标记，Win 32 API也需要它：

```
SELECT GET_FILESTREAM_TRANSACTION_CONTEXT()
```

查询返回：

```
0x57773034AFA62746966EE30DAE70B344
```

在得到这个信息后，应用程序可以使用OpenSQLFileStream API和路径及事务标记一起来执行ReadFile和WriteFile函数，然后关闭文件的句柄。

82

在应用程序结束之后，可以回滚或提交这个事务：

```
COMMIT TRAN
```

如果希望删除文件，可以将列值设为NULL：

```
UPDATE dbo.BookPhoto
SET BookPhotoFile = NULL
WHERE BookPhotoNM = 'SQL Server 2008 Transact-SQL Recipes cover'
```

你可能不会马上看到文件系统下的底层文件被删除，不过它最终会被垃圾收集进程删除掉。

解析

在这个技巧中，演示了如何在SQL Server 2008中使用varbinary(max)数据类型的FILESTREAM属性将非结构化数据存储在文件系统中。它启用了SQL Server在SQL Server内控制事务和可恢复性（使用BACKUP备份文件，RESTORE来恢复）的功能，使用Win 32 API也会带来高速的流性能。

在这个技巧中，先检查SQL Server实例是否启用了FILESTREAM。在这之后，创建了新的数据库，指定了FILESTREAM文件组的位置和文件名称（实际上是路径而不是文件）：

```
...
  FILEGROUP FS_PhotoRepository CONTAINS FILESTREAM
  (NAME = 'FG_PhotoRepository',
   FILENAME = N'C:\Apress\FILESTREAM')
...
```

要记住最后的文件夹之前的路径必须存在，而最后引用的文件夹不能存在。例如，C:\Apress存在于服务器中，但是，FILESTREAM不可以在数据库创建之前存在。

在创建数据库之后，创建了用于存储书籍封面图片的表。为BookPhotoFile列指定了varbinary(max)类型，随后是FILESTREAM属性：

```
...
    BookPhotoFile varbinary(max) FILESTREAM)
...
```

在关闭FILESTREAM属性之后，所有varbinary数据都会存储在数据库数据文件中，而不会存储在文件系统中。列最大尺寸限制在2GB之内。

然后，将一个新行插入到表中，它保存了*SQL Server 2008 Transact-SQL Recipes*的封面的BMP文件。使用在前一技巧中演示过的OPENROWSET技术生成varbinary(max)值：

```
INSERT dbo.BookPhoto
(BookPhotoID, BookPhotoNM, BookPhotoFile)
SELECT NEWID(),
       'SQL Server 2008 Transact-SQL Recipes cover',
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
      BulkColumn
FROM OPENROWSET(BULK
'C:\Apress\TSQL2008Recipes.bmp', SINGLE_BLOB) AS x
```

从应用程序的角度看，要利用使用Win 32 API的流能力，需要一些信息。首先打开一个新的事务：

```
BEGIN TRAN
```

在这之后，使用PathName()方法引用到存储文件的路径名称：

```
SELECT BookPhotoFile.PathName()
...
```

这个函数返回路径作为标记，之后应用程序可以使用它获取Win32句柄，并对值实施操作。

下一步，调用GET_FILESTREAM_TRANSACTION_CONTEXT函数返回表示当前会话事务上下文的标记。

```
SELECT GET_FILESTREAM_TRANSACTION_CONTEXT()
```

这是应用程序用来将文件系统操作绑定到实际事务所使用的标记。

在这之后，演示了如何“删除”文件：通过更新之前增加的指定行的BookPhotoFile列为NULL。要记住删除实际的行也可以达到同样的目的（在文件系统中删除文件）。

2.2.6 “就地”赋值并修改数据库值

除了标准的等于（=）运算符，SQL Server 2008还引入了新的、既可以赋值又可以修改数据值的复合赋值运算符。这些运算符与你在C和Java语言中看到的相似。新的赋值运算符包括以下几个：

- +=（加，赋值）；
- -=（减，赋值）；
- *=（乘，赋值）；
- /=（除，赋值）；
- |=（位或，赋值）；
- ^=（位异或，赋值）；
- &=（位与，赋值）；
- %=（取模，赋值）；

这个技巧将演示使用赋值运算符修改基础薪金。下面先创建一个新表并填充一些数据值。

```
USE AdventureWorks
GO

CREATE TABLE HumanResources.EmployeePayScale
     (EmployeePayScaleID int NOT NULL PRIMARY KEY IDENTITY(1,1),
     BasePayAMT numeric(9,2) NOT NULL,
     ModifiedDate datetime NOT NULL DEFAULT GETDATE())
GO

-- 使用新的多行插入功能
INSERT HumanResources.EmployeePayScale
(BasePayAMT)
VALUES
      (30000.00),
      (40000.00),
      (50000.00),
      (60000.00)
```

下一步复查指定的薪级行的初始值：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SELECT  BasePayAMT
FROM    HumanResources.EmployeePayScale
WHERE   EmployeePayScaleID = 4
```

查询返回：

```
BasePayAMT
60000.00
```

在SQL Server 2008之前，如果希望在UPDATE中修改一个值，且该修改基于行中现在的值，则需要做如下操作：

```
UPDATE  HumanResources.EmployeePayScale
SET BasePayAMT  =  BasePayAMT + 10000
WHERE   EmployeePayScaleID = 4
```

查询这一行，发现基础薪金增加了10 000：

```
SELECT  BasePayAMT
FROM    HumanResources.EmployeePayScale
WHERE   EmployeePayScaleID = 4
```

查询返回：

```
BasePayAMT
70000.00
```

现在开始对赋值运算符做实验。这个新特性可以简化代码——就地赋值，而不需要在值表达式中对另一列引用。

在这个示例中，基础薪金再一次增加了10 000元：

```
UPDATE  HumanResources.EmployeePayScale
SET BasePayAMT += 10000
WHERE   EmployeePayScaleID = 4

SELECT  BasePayAMT
FROM    HumanResources.EmployeePayScale
WHERE   EmployeePayScaleID = 4
```

查询返回：

```
BasePayAMT
80000.00
```

下一步，基础薪金被乘以2：

```
UPDATE  HumanResources.EmployeePayScale
SET BasePayAMT *= 2
WHERE   EmployeePayScaleID = 4

SELECT  BasePayAMT
FROM    HumanResources.EmployeePayScale
WHERE   EmployeePayScaleID = 4
```

85

查询返回：

```
BasePayAMT
160000.00
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

复合赋值运算符帮助你用最少的代码量修改值。在这个技巧中，演示了使用加赋值运算符：

```
SET BasePayAMT += 10000
```

以及乘/赋值运算符：

```
SET BasePayAMT *= 2
```

表达式在左端指定了要修改的列名，随后是赋值运算符，然后是相应的运算符要使用的数据值。要记住这个功能不仅限于在UPDATE语句中使用，也可以使用这个新功能对变量进行赋值。

2.3 DELETE

DELETE的简单语法如下：

```
DELETE [FROM] table_or_view_name
WHERE search_condition
```

表2-7描述了这个命令的参数。

表2-7 DELETE命令参数

参 数	描 述
table_or_view_name	将要删除的行所属的表或可更新视图的名字
search_condition	WHERE子句中的搜索条件定义了哪些行会从表或可更新视图中删除

2.3.1 删除行

在这个技巧中，将介绍如何使用DELETE语句从表中移除一行或者多行。首先，做一个示例表并填充上行：

```
SELECT *
INTO Production.Example_ProductProductPhoto
FROM Production.ProductProductPhoto
```

这个查询返回：

```
(504行受影响)
```

然后，删除表中的所有行：

```
DELETE Production.Example_ProductProductPhoto
```

这个查询返回：

```
(504行受影响)
```

下一个示例演示使用带WHERE子句的DELETE。假设两个表的关系键丢失了，用户能从主键表删除数据但是不能从相关的外键表删除数据（有关主键和外键的更多信息可以参阅第4章）。示例中只有那些在Product表中失去了相应条目的行才会从示例的产品图片表中删除，在这个示例中，没有行满足这个条件：

```
--重新填充Example_ProductProductPhoto表
INSERT Production.Example_ProductProductPhoto
SELECT *
FROM Production.ProductProductPhoto
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DELETE Production.Example_ProductProductPhoto
WHERE ProductID NOT IN
    (SELECT ProductID
     FROM Production.Product)
```

第三个示例演示了和前面一个示例相同的功能，只不过使用FROM子句重写了DELETE，而不是子查询：

```
DELETE Production.ProductProductPhoto
FROM Production.Example_ProductProductPhoto ppp
LEFT OUTER JOIN Production.Product p ON
  ppp.ProductID = p.ProductID
WHERE p.ProductID IS NULL
```

解析

在技巧的第一个示例中，Example_ProductProductPhoto表的所有行都被删除了：

```
DELETE Production.Example_ProductProductPhoto
```

这是因为没有使用WHERE子句来限定哪些行需要删除。

在第二个示例中，我们使用了WHERE子句，根据对其他表的子查询指定了需要删除的行：

```
WHERE ProductID NOT IN
    (SELECT ProductID
     FROM Production.Product)
```

第三个示例使用LEFT OUTER JOIN来代替子查询，它联结了两个表的ProductID：

```
FROM Production.Example_ProductProductPhoto ppp
LEFT OUTER JOIN Production.Product p ON
  ppp.ProductID = p.ProductID
```

因为要从Production.ProductProductPhoto删除的对象就是FROM子句中的对象，而且在FROM子句中只有一个对表的引用，所以可以假设FROM和WHERE子句识别的行是同一行——它会关联到从Production.ProductProductPhoto表删除的行。

87

因为使用了LEFT OUTER JOIN，所以如果某行在左表和右表中没有找到匹配，右表对于没有匹配的ProductID会给出NULL值。因此，要找出Production.Example_ProductProductPhoto表中ProductID不能匹配Production.Product表的行，可以如下限定Production.Product：

```
WHERE p.ProductID IS NULL
```

任何没有匹配Production.Product表的行会从Production.Example_ProductProductPhoto表中删除。

2.3.2　截断表

在这个技巧中，将会介绍如何从表中删除行，但记录最少的日志（因此就算是非常大的表也会删除得很快）。通常，如果希望完全记录日志就应该使用DELETE操作。然而，如果是用于测试或者丢弃数据，这是一个快速移除数据的技术。最小日志意味着有多少恢复信息写入了数据库的事务日志（参阅第22章）。

语法如下：

```
TRUNCATE TABLE table_name
```

这个命令只接受需要截断的表名。由于它总是从表中移除所有行，所以没有FROM和WHERE子句，如下面技巧所演示的那样：

```
--首先填充示例数据
SELECT *
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
INTO Sales.Example_Store
FROM Sales.Store

--然后，截断示例表中的所有行
TRUNCATE TABLE Sales.Example_Store
```

接下来查询表中的行数：

```
SELECT COUNT(*)
FROM Sales.Example_Store
```

这个查询返回：

```
0
```

解析

TRUNCATE TABLE语句和DELETE语句差不多，用于从表中删除行。TRUNCATE TABLE删除行比DELETE快，因为它记录很少的日志。但是和DELETE不同的是，TRUNCATE TABLE必须删除表中的所有行（没有WHERE子句）。

尽管TRUNCATE TABLE能更快地删除行；但是如果表中的列具有外键约束（有关外键的更多信息参阅第4章），或者表使用事务复制或合并复制发布，或者表参与了一个索引的视图（更多信息请参阅第4章），则不能使用它。还要记住，如果表有IDENTITY列，列会被重置为列定义的种子值（如果没有显式设置种子，则重置成1）。

2.4 高级数据修改技术

随后两个技巧将演示更高级的数据修改技术。特别地，我将演示如何通过将数据分成一些更小的集合来提升数据修改的吞吐量。

我也将演示SQL Server 2008的新MERGE命令，使用它可以将改变高效地应用到基于表源中的数据的目标表，而不用指定多个DML语句。

2.4.1 使用 TOP 分块修改数据

在第1章中已经介绍过TOP的使用。TOP同样也能用于DELETE、INSERT或者UPDATE语句。这个技巧会进一步演示使用TOP来对数据修改分块操作，也就是不在一个语句调用中执行一个非常大的操作，而是把修改分成小块，这样能提高性能并且改善大数据量、大访问量的表的数据库并发性。这种技术经常用于大数据量的报表或者数据仓库应用程序。

大量独立的更新会导致数据库事务日志快速增长。如果分块操作的话，每块完成后提交，这样SQL Server就有可能重用事务日志空间。除了事务日志空间，对于一个大量数据的更新，如果查询必须取消，那么可能不得不等待很长一段时间让事务回滚。而如果分成小块，你就能很快继续你的更新。同样，分块能让修改表有更多并发性，让用户可以进行查询，而不是要等所有的修改都完成。

在这个技巧中，将会介绍如何通过多次执行来分块修改数据，而不是通在一个大的事务中执行整个结果集来修改数据。首先，我们为这个技巧创建一个示例删除表：

```
USE AdventureWorks
GO

SELECT *
INTO Production.Example_BillOfMaterials
FROM Production.BillOfMaterials
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

然后，表中的所有行会以500行为一块进行删除：

```
WHILE (SELECT COUNT(*)FROM Production.Example_BillOfMaterials)> 0
BEGIN

  DELETE TOP(500)
  FROM Production.Example_BillOfMaterials

END
```

这个查询返回：

```
(500行受影响)
(500行受影响)
(500行受影响)
(500行受影响)
(500行受影响)
(179行受影响)
```

89

解析

在这个示例中，我使用了WHILE条件，只要表中的总行数大于0则不断执行DELETE（有关WHILE的更多信息请参阅第9章）：

```
WHILE (SELECT COUNT(*)FROM Production.Example_BillOfMaterials)> 0
BEGIN
```

然后就是DELETE，后面跟随TOP子句并且在圆括号内限定了行数：

```
DELETE TOP(500)
FROM Production.BillOfMaterials
```

这个技巧没有使用WHERE子句，所有没有应用过滤，并且表中所有的行都被删除——但是一次500行。一旦WHILE条件不再计算为TRUE，循环就结束了。执行后，每一批操作影响的行数都会被显示。一开始的5批删除了500行，最后一批删除了余下的179行。

这种“分块”的方法也能用于INSERT和UPDATE。对于INSERT和UPDATE，TOP子句就跟在INSERT和UPDATE关键字后面，例如：

```
INSERT TOP(100)
...

UPDATE TOP(25)
...
```

TOP的这种扩展功能为表的大批数据修改增加了一项新技术。通过减小大型修改操作的大小，以便减少操作时锁的时间，从而提升数据库的并发性（让其他会话离开小窗口），这也有助于控制事务日志的大小（多次提交，而不是一个大型事务只提交一次）[①]。

2.4.2　在一条语句中执行 INSERT、UPDATE 和 DELETE

SQL Server 2008引入了MERGE命令来将改变高效应用到基于表源中数据的目标表。如果你曾经不得不载入并不停地修改关系型数据仓库，或操作的数据存储基于从外部数据源传入的数据，你会发现这

① 该段的意思是，新的TOP技术可以把一步完成的大批数据修改操作改变为多个小的操作，产生多个提交，不会长时间锁表，以此来提升数据库的并发性。——译者注

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

个技术与以前的方法相比有很大的改进。

你可以不用创建多条数据修改语句，而只使用一个MERGE到目标或源表，定义当搜索条件找到匹配、当目标表没有匹配时或当源表没有匹配时执行什么操作。基于这些匹配条件，你可以指定是否执行DELETE、INSERT或UPDATE操作（再次说明是在同一语句中）。

这个技巧演示基于分期表的数据（可能分期数据是从外部数据源而来）将修改应用到一个产品表。下面先创建一个产品表，告诉我某公司住房单位是否可租赁。如果IsRentedIND为0，则这个单位是不可用的，如果它为1，单位就是可用的：

```
CREATE TABLE HumanResources.CorporateHousing
     (CorporateHousingID int NOT NULL PRIMARY KEY IDENTITY(1,1),
      UnitNBR int NOT NULL,
      IsRentedIND bit NOT NULL,
      ModifiedDate datetime NOT NULL DEFAULT GETDATE())
GO

-- 插入现有的单位
INSERT HumanResources.CorporateHousing
(UnitNBR, IsRentedIND)
VALUES
(1, 0),
(24, 1),
(39, 0),
(54, 1)
```

在这个情景中，我定期接收告知我公司住房单位租赁状态改变的资料。单位可以在已租和未租状态中切换。在合同签字后可以增加新的单位，在合同修改或变更需求后可以删除现有的单位。因此在这个技巧中，将创建从外部数据源接收当前公司住房单位状态的分期表。我也会将最新的信息添加到表中：

```
CREATE TABLE dbo.StagingCorporateHousing
     (UnitNBR int NOT NULL,
      IsRentedIND bit NOT NULL)
GO

INSERT dbo.StagingCorporateHousing
(UnitNBR, IsRentedIND)
VALUES
-- UnitNBR "1"不再存在
(24, 0), -- UnitNBR 24 改变了租赁状态
(39, 1), -- UnitNBR 39 也一样。
(54, 0), -- UnitNBR 54 改变了状态。
(92, 1) -- UnitNBR 92 是新的单位，还没有成为产品 .
```

现在我的目标是修改目标产品表，让它可以反映数据源中最新的数据。如果在分期表中存在新的单位，我希望增加它。如果单位编号存在于产品表中，而不在分期表中，我希望删除这一行。如果单位编号同时存在于分期表和产品表中，但租赁状态不同，我希望更新产品（目标）表来反映变化。

在修改之前，先查看产品的值。

```
-- 在MERGE之前
SELECT CorporateHousingID, UnitNBR, IsRentedIND
FROM HumanResources.CorporateHousing
```

查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CorporateHousingID    UnitNBR    IsRentedIND
1                     1          0
2                     24         1
3                     39         0
4                     54         1
```

下一步，将根据业务需求来修改产品表：

```
MERGE INTO HumanResources.CorporateHousing p
USING dbo.StagingCorporateHousing s
ON p.UnitNBR = s.UnitNBR
WHEN MATCHED AND s.IsRentedIND <> p.IsRentedIND THEN
UPDATE SET IsRentedIND = s.IsRentedIND
WHEN NOT MATCHED BY TARGET THEN
INSERT (UnitNBR, IsRentedIND) VALUES (s.UnitNBR, s.IsRentedIND)
WHEN NOT MATCHED BY SOURCE THEN
DELETE;
```


查询返回：

```
(5行受影响)
```

然后检查修改产品表“之后的”结果：

```
-- 在MERGE之后
SELECT CorporateHousingID, UnitNBR, IsRentedIND
FROM HumanResources.CorporateHousing
```

查询返回：

```
CorporateHousingID    UnitNBR    IsRentedIND
2                     24         0
3                     39         1
4                     54         0
5                     92         1
```

解析

在这个技巧中，我演示了如何使用MERGE语句应用INSERT/UPDATE/DELETE修改。MERGE命令允许我基于审定源分期表的表达式修改目标表。

在MERGE命令的第一行，指定了我要应用数据修改的目标表：

```
MERGE INTO HumanResources.CorporateHousing p
```

在第二行，指定了用来与目标表比较数据的数据源。这个源也可以基于派生表或链接服务器表：

```
USING dbo.StagingCorporateHousing s
```

下一步，定义如何联结这两个数据源。在这个例子中，使用了数据的自然键。这个自然键唯一标识既在源表又在目标表中的那些行：

```
ON p.UnitNBR = s.UnitNBR
```

下一步，通过指定WHEN MATCHED定义当单位编号匹配时会发生什么。我还增加了一个额外的搜索条件——它指出如果租金指标不匹配，租金指标就会修正到匹配分期数据：

```
WHEN MATCHED AND s.IsRentedIND <> p.IsRentedIND THEN
   UPDATE SET IsRentedIND = s.IsRentedIND
```

下一步，评估当源表和目标表没有匹配时会发生什么——例如，如果源表存在单位编号为92的值，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

而目标表不存在这样的行。当发生这种情况后，则增加缺失的行到目标表：

```
WHEN NOT MATCHED BY TARGET THEN
INSERT (UnitNBR, IsRentedIND) VALUES (s.UnitNBR, s.IsRentedIND)
```


最后，如果存在于目标表的行在源分期表中不存在，则从目标产品表中删除该行：

```
WHEN NOT MATCHED BY SOURCE THEN
DELETE;
```

注意，这是需使用分号进行终结的命令之一。你也会注意到，不是每个MERGE命令都需要指定INSERT、UPDATE或DELETE。你可能决定只希望增加新的行并更新现有的行。或者你也可能这样决定，希望通过更新一个标记“逻辑地”从产品表中删除行，而不是真正从表中删除它。

使用MERGE将数据修改应用到目标表，需要的代码量比早期版本的更少。当持续地应用数据修改时也会带来一些性能上的益处，因为这样只需要一次从源到目标数据的传递，而不是为每个修改类型做多次传递。

2.5 捕获和跟踪数据修改的改变

本章的最后几个技巧将演示如何捕获和跟踪数据修改活动。

在第一个技巧中，将展示如何使用OUTPUT子句显示受INSERT、UPDATE或DELETE操作影响的行。在这之后，将演示两个SQL Server 2008新特性：变更数据捕获（Change Data Capture，CDC）和更改跟踪。

CDC只有很少的性能开销，并且可以用来持续地更新其他数据源，例如，将OLTP数据库中的变更迁移到数据仓库数据库。

CDC用于异步跟踪数据存储和数据仓库持续的数据变更，同时也提供了侦测数据中间变更的能力；而更改跟踪是异步进程，它是DML操作本身（INSERT、UPDATE、DELETE）事务的一部分，用来以最小的磁盘存储开销侦测网络行变更。

2.5.1 返回数据修改语句影响的行

在这个技巧中，将展示如何使用OUTPUT子句来返回那些被INSERT、UPDATE或DELETE操作影响的行信息（也可以对MERGE进行捕获）。在第一个示例中，UPDATE语句修改了指定产品的名称。然后使用OUTPUT返回原始的和更新后的列名称信息：

```
DECLARE @ProductChanges TABLE
  (DeletedName nvarchar(50),
   InsertedName nvarchar(50))

UPDATE Production.Product
SET Name = 'HL Spindle/Axle XYZ'
OUTPUT DELETED.Name,
     INSERTED.Name
 INTO @ProductChanges
WHERE ProductID = 524
SELECT DeletedName,
    InsertedName
FROM @ProductChanges
```


这个查询返回：

```
DeletedName       InsertedName
HL Spindle/Axle   HL Spindle/Axle XYZ
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下一个示例为DELETE操作使用OUTPUT。首先，创建一个保存数据的表：

```
SELECT *
INTO Sales.Example_SalesTaxRate
FROM Sales.SalesTaxRate
```

接下来的一段代码创建用于保存数据的表变量，从表中删除行，然后从表变量中查询哪些行被删除了：

```
DECLARE @SalesTaxRate TABLE(
  [SalesTaxRateID] [int] NOT NULL,
  [StateProvinceID] [int] NOT NULL,
  [TaxType] [tinyint] NOT NULL,
  [TaxRate] [smallmoney] NOT NULL,
  [Name] [dbo]. [Name] NOT NULL,
  [rowguid] [uniqueidentifier] ,
  [ModifiedDate] [datetime] NOT NULL )

DELETE Sales.Example_SalesTaxRate
OUTPUT DELETED.*
INTO @SalesTaxRate

SELECT SalesTaxRateID,
    Name
FROM @SalesTaxRate
```

这段查询返回下面的部分结果：

```
SalesTaxRateID    Name
1                 Canadian GST + Alberta Provincial Tax
2                 Canadian GST + Ontario Provincial Tax
3                 Canadian GST + Quebec Provincial Tax
4                 Canadian GST
...
27                Washington State Sales Tax
28                Taxable Supply
29                Germany Output Tax
30                France Output Tax
31                United Kingdom Output Tax

(29行受影响)
```

在第三个示例中，将演示带OUTPUT的INSERT。向表中插入一个新行，而且这个操作被捕获到一个表变量中：

94

```
DECLARE @NewDepartment TABLE
  (DepartmentID smallint NOT NULL,
   Name nvarchar(50) NOT NULL,
   GroupName nvarchar(50) NOT NULL,
   ModifiedDate datetime NOT NULL)

INSERT HumanResources.Department
(Name, GroupName)
OUTPUT INSERTED.*
INTO @NewDepartment
VALUES ('Accounts Receivable', 'Accounting')

SELECT DepartmentID,
    ModifiedDate
FROM @NewDepartment
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回：

```
DepartmentID    ModifiedDate
18              2007-09-15 08:38:28.833
```


解析

第一个示例使用临时表变量来保存OUTPUT结果（有关临时表变量的更多信息请参阅第4章）：

```
DECLARE @ProductChanges TABLE
  (DeletedName nvarchar(50),
   InsertedName nvarchar(50))
```

然后，UPDATE的第一部分把产品名修改成HL Spindle/Axle XYZ：

```
UPDATE Production.Product
SET Name = 'HL Spindle/Axle XYZ'
```

在SET子句之后和WHERE子句之前，OUTPUT定义了要返回哪些列：

```
OUTPUT DELETED.Name,
     INSERTED.Name
```

和DML触发器（第12章会讲到）差不多，OUTPUT使用两个“虚拟”表INSERTED和DELETED，它们保存了已更新表的原始值和修改后的值。INSERTED和DELETED虚拟表与被修改的表具有同样的列名——在本例中返回原始的名字（DELETED.Name）和新的名字（INSERTED.Name）。

使用INTO把OUTPUT的值放入临时表变量中，INTO之后跟随的是表名：

```
INTO @ProductChanges
```

UPDATE查询限定了只有ProductID为524的产品才会被修改成新的名字：

```
WHERE ProductID = 524
```

更新之后，我们对@ProductChanges临时表变量执行查询来显示修改前/后的数据：

```
SELECT DeletedName,
  InsertedName
FROM @ProductChanges
```

95

使用了OUTPUT的DELETE和INSERT示例与第一个示例相比有一些变化，OUTPUT把删除的行（对于DELETE操作）或插入的行（对于INSERT操作）放入表变量。

2.5.2 异步捕获表数据修改

SQL Server 2008为异步跟踪所有发生在用户表上的数据修改提供了内建的方法，而不需要编写自定义触发器或查询。变更数据捕获拥有最小性能开销，可以用于其他数据源的持续更新，例如，将OLTP数据库中的数据变更迁移到数据仓库数据库。下一组技巧将演示如何使用这个新功能。

首先，创建用来演示这个功能的新的数据库。

```
IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'TSQLRecipe_CDC_Demo')

BEGIN
     CREATE DATABASE TSQLRecipe_CDC_Demo
END
GO
```

在第一个技巧中，将演示增加CDC到TSQLRecipe_CDC_Demo数据库的表中。第一步是验证数据库是

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

否启用了变更数据捕获：

```
SELECT is_cdc_enabled
FROM sys.databases
WHERE name = 'TSQLRecipe_CDC_Demo'
```

查询返回：

```
is_cdc_enabled
0
```

使用各种存储过程来配置和管理变更数据捕获。为了启用数据库，下面将在TSQLRecipe_CDC_Demo数据库的上下文中执行sys.dp_cdc_enable_db存储过程：

```
USE TSQLRecipe_CDC_Demo
GO

EXEC sys.sp_cdc_enable_db
GO
```

查询返回：

```
命令已成功完成。
```

下一步，将重新验证变更数据捕获是否已启用：

```
SELECT is_cdc_enabled
FROM sys.databases
WHERE name = 'TSQLRecipe_CDC_Demo'
```

查询返回：

```
is_cdc_enabled
1
```


现在变更数据捕获已启用，我可以在数据库中通过使用sys.sp_cdc_enable_table系统存储过程对表进行变更捕获。这个存储过程的参数在表2-8中描述。

表2-8　sp_cdc_enable_table的参数

参　数	描　述
@source_schema	这个参数定义了对象的架构
@source_name	这个参数指定了表名称
@role_name	这个选项允许你选择拥有访问CDC数据权限的用户定义角色的名称
@capture_instance	一个表最多可以指定两个捕获实例。当你计划改变已被CDC捕获的表的架构时，这就派的上用场。你可以在不影响原始CDC的情况下改变架构（除非它是一个数据类型变更），创建新的捕获实例，在两个表中跟踪变更，然后在你确认新的架构捕获适合你的需求后删除原始捕获实例。如果你不指定名称，默认值为schema_source
@supports_net_changes	当启用时，这个选项允许你只显示在LSN范围内最近的数据变更。这个选项需要表定义一个主键。如果没有定义主键，也可以在@index_name选项中指定一个唯一键
@index_name	这个参数允许在CDC使用的表不存在主键的情况下指定唯一键
@captured_column_list	如果你对跟踪所有的列变更无兴趣，这个选项允许你减小列表

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

参 数	描 述
@filegroup_name	这个选项允许你指定CDC数据存储在哪里。对于大型数据集，通过对不同的文件组进行分隔可能会带来更好的管理性及性能
@partition_switch	这个参数用TRUE或FALSE来指定是否允许对CDC表使用ALTER TABLE...SWITCH PARTITION命令（默认为FALSE）

在这个技巧中，将跟踪下面的新表所有的变更：

```
USE TSQLRecipe_CDC_Demo
GO

CREATE TABLE dbo.Equipment
   (EquipmentID int NOT NULL PRIMARY KEY IDENTITY(1,1),
    EquipmentDESC varchar(100) NOT NULL,
    LocationID int NOT NULL)
GO
```

我想要捕获所有发生在行的变更，并且只返回行的净变更。对于其他选项，都将使用默认值：

```
EXEC sys.sp_cdc_enable_table
    @source_schema = 'dbo',
    @source_name = 'Equipment',
    @role_name = NULL,
    @capture_instance = NULL,
    @supports_net_changes = 1,
    @index_name = NULL,
    @captured_column_list = NULL,
    @filegroup_name = default
```


这个过程调用的结果表示创建了两个SQL Server 代理作业（SQL Server代理必须是运行的）：

```
作业'cdc.TSQLRecipe_CDC_Demo_capture' 已成功启动。
作业'cdc.TSQLRecipe_CDC_Demo_cleanup' 已成功启动。
```

为每个为表启用CDC的数据库创建了捕获和清除两个作业。

提示 如果CDC已经在同一数据库的表中启用，作业将不会重新创建。

可以通过执行下面的查询确认这个表已经被跟踪：

```
SELECT is_tracked_by_cdc
FROM sys.tables
WHERE name = 'Equipment' and
   schema_id = SCHEMA_ID('dbo')
```

查询返回：

```
is_tracked_by_cdc
1
```

也可以使用sys.sp_cdc_help_change_data_capture存储过程来验证新配置的捕获实例的设置：

```
EXEC sys.sp_cdc_help_change_data_capture 'dbo', 'Equipment'
```

查询返回如下的结果集（为了便于查看，结果以名称/值的格式表示）：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
source_schema           dbo
source_table            Equipment
capture_instance        dbo_ Equipment
object_id               357576312
source_object_id        293576084
start_lsn               NULL
end_lsn                 NULL
supports_net_changes    1
has_drop_pending        NULL
role_name               NULL
index_name              PK__Equipmen__344745994707859D
filegroup_name          NULL
create_date               2008-03-16 09:27:52.990
index_column_list       [EquipmentID]
captured_column_list    [EquipmentID], [EquipmentDESC], [LocationID]
```

98

解析

在这个技巧中，首先通过使用sp_cdc_enable_db启用数据库CDC功能。在后台，启用数据库的CDC创建了名为cdc的新架构和数据库中的一些新表，表2-9中对这些表进行了详细说明。不需要直接查询这些表，因为有系统存储过程和函数可以以更清晰的格式返回相同的数据。

表2-9　CDC系统表

表	描　述
cdc.captured_columns	返回指定捕获实例跟踪的列
cdc.change_tables	返回当启用表的CDC时创建的表。使用sys.sp_cdc_help_change_data_capture来查询这个信息比直接查询这个表更好
cdc.ddl_history	在启用CDE后，为每个作用于表的DDL变更返回行，可以使用sys.sp_cdc_get_ddl_history代替直接查询这个表
cdc.index_columns	返回与启用CDC的表相关的索引列。查询sys.sp_cdc_help_change_data_capture获取这个信息比直接查询这个表要好
Cdc_lsn_time_mapping	帮助你在日志序列号到事务的开始和结束时间之间建立映射。再次说明，避免直接查询这个表，而要使用sys.fn_cdc_map_lsn_to_time和sys.fn_cdc_map_time_to_lsn函数

我将复习如何在随后的技巧中使用更普遍使用的函数和过程。

在数据库中启用CDC之后，使用sp_cdc_enable_table过程增加了对数据库中用户表的CDC跟踪。我指定了架构、名称和净变更标记。所有其他选项保留为默认值。

在执行sp_cdc_enable_table之后，因为数据库中第一次启用了源表，所以创建了两个新SQL代理作业。一个作业名称是cdc.TSQLRecipe_CDC_Demo_capture。这个作业的职责是使用复制日志读取器技术捕获发生的变更，并且在SQL Server启动并连续运行时被预定为自动启动。第二个作业是cdc.TSQLRecipe_CDC_Demo_cleanup，默认预定每天上午2点运行，并且默认清除存放3天（72小时）以上的数据。

执行sys.sp_cdc_help_change_data_capture可以验证捕获实例的各种设置，包括净变更的支持、跟踪列、创建日期和用于确定行的唯一性的主键。

为表启用CDC也导致在CDC架构中创建了一个新表。在这个例子中，自动创建了名为cdc.dbo_Equipment_CT的新表。这个表与基础表拥有相同的列，并增加了跟踪LSN、操作和更新列信息的5个额外的列。不应该直接查询这个表，而要使用下一技巧演示的函数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

2.5.3 从 CDC 表查询所有变更

现在，已经为数据库启用了CDC，并且为表创建了变更捕获实例，为了演示这个功能，将继续使这个表发生变更：

99


```
USE TSQLRecipe_CDC_Demo
GO

INSERT dbo.Equipment
(EquipmentDESC, LocationID)
VALUES ('Projector A', 22)

INSERT dbo.Equipment
(EquipmentDESC, LocationID)
VALUES ('HR File Cabinet', 3)

UPDATE dbo.Equipment
SET EquipmentDESC = 'HR File Cabinet 1'
WHERE EquipmentID = 2

DELETE dbo.Equipment
WHERE EquipmentID = 1
```

在生成变更之后，现在可以使用CDC函数查看变更历史了。在日志序列号（Log sequence number，LSN）级别跟踪数据变更。LSN是在事务日志中唯一标识活动的一个记录。

现在要获取我希望生成变更的时间范围的最小和最大的LSN值。这里使用sys.fn_cdc_map_time_to_lsn函数来确定LSN，这个函数有两个输入参数：关系运算符和跟踪时间（还可以使用其他方法来确定LSN，本章稍后会演示）。关系运算符如下：

- smallest greater than;
- smallest greater than or equal;
- largest less than;
- largest less than or equal。

帮助确定相关的LSN值这些运算符用来和更改跟踪时间结合使用。对于这个技巧，我希望最小和最大LSN值在两个时间之间：

```
SELECT sys.fn_cdc_map_time_to_lsn
( 'smallest greater than or equal' , '2008-03-16 09:34:11') as BeginLSN

SELECT sys.fn_cdc_map_time_to_lsn
( 'largest less than or equal' , '2008-03-16 23:59:59') as EndLSN
```

查询返回如下结果（如果你跟着执行示例程序，实际的LSN将会不同）：

```
BeginLSN
0x0000001C000001020001

(1行受影响)

EndLSN
0x0000001C000001570001

(1行受影响)
```

现在有了侦测发生在希望的时间范围内的变更的LSN边界。

100

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下一个决定是希望看到所有变更还是只看到净变更。可以调用在前一查询中演示过的一个函数，生成LSN边界并把它们存入变量，供cdc.fn_cdc_get_all_changes_dbo_Equipment函数使用。如同那个函数名称所暗示的，将首先演示显示所有变更：

```
DECLARE @FromLSN varbinary(10) =
sys.fn_cdc_map_time_to_lsn
( 'smallest greater than or equal' , '2008-03-16 09:34:11')

DECLARE @ToLSN varbinary(10) =
sys.fn_cdc_map_time_to_lsn
( 'largest less than or equal' , '2008-03-16 23:59:59')

SELECT
   __$operation,
   __$update_mask,
   EquipmentID,
   EquipmentDESC,
   LocationID
FROM cdc.fn_cdc_get_all_changes_dbo_Equipment
   (@FromLSN, @ToLSN, 'all')
```

查询返回如下的结果集：

__$operation	__$update_mask	EquipmentID	EquipmentDESC	LocationID
2	0x07	1	Projector A	22
2	0x07	2	HR File Cabinet	3
4	0x02	2	HR File Cabinet 1	3
1	0x07	1	Projector A	22

这个结果集展示了表发生的所有修改。注意函数的名称cdc.fn_cdc_get_all_changes_dbo_Equipment，它是根据CDC实例要捕获的源表的名称命名的。也要注意到__$operation和__$update_mask的值。__$operation值的解释如下：

- 1为删除；
- 2为插入；
- 3为更新的行“之前”的版本（使用all update old选项来查看，我在前面的查询中没有使用这个）；
- 4为更新的行“之后”的版本。

更新掩码使用位来表示某一操作修改的捕获列。我会单独用一个技巧演示如何翻译这些值。

推进到这个技巧，也使用all update old选项来显示更新行在修改之前的值。我还增加逻辑来翻译在结果集中看到的操作类型的值。例如：

```
DECLARE @FromLSN varbinary(10) =
  sys.fn_cdc_map_time_to_lsn
   ( 'smallest greater than or equal' , '2008-03-16 09:34:11')

DECLARE @ToLSN varbinary(10) =
  sys.fn_cdc_map_time_to_lsn
   ( 'largest less than or equal' , '2008-03-16 23:59:59')

SELECT
   CASE __$operation
      WHEN 1 THEN 'DELETE'
      WHEN 2 THEN 'INSERT'
      WHEN 3 THEN 'Before UPDATE'
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        WHEN 4 THEN 'After UPDATE'
    END Operation,
    __$update_mask,
    EquipmentID,
    EquipmentDESC,
    LocationID
FROM cdc.fn_cdc_get_all_changes_dbo_Equipment
    (@FromLSN, @ToLSN, 'all update old')
```

查询返回：

Operation	__$update_mask	EquipmentID	EquipmentDESC	LocationID
INSERT	0x07	1	Projector A	22
INSERT	0x07	2	HR File Cabinet	3
Before UPDATE	0x02	2	HR File Cabinet	3
After UPDATE	0x02	2	HR File Cabinet 1	3
DELETE	0x07	1	Projector A	22

解析

在这个技巧中，修改了CDC跟踪的表。因为底层的CDC数据实际通过LSN来跟踪，所以需要将最小/最大时间范围转换为包含要查找的数据变更的最小和最大LSN。使用sys.fn_cdc_map_time_to_lsn来完成这个转换。

提示 sys.fn_cdc_map_lsn_to_time函数可以将LSN转换为时间值。

下一步，执行cdc.fn_cdc_get_all_changes_dbo_Equipment函数，使用它可以返回传入的LSN范围内发生的所有变更：

```
SELECT
    __$operation,
    __$update_mask,
    EquipmentID,
    EquipmentDESC,
    LocationID
FROM cdc.fn_cdc_get_all_changes_dbo_Equipment
    (@FromLSN, @ToLSN, 'all')
```

对于正在进行的持续的载入，可能有必要为每个载入都存储开始和结束的LSN值，然后使用sys.fn_cdc_increment_lsn函数来将旧的上限LSN值作为下一轮载入的下限LSN值（我将在稍后的技巧中进行演示）。

在本技巧的最后一个示例中，使用了all update old参数来返回UPDATE语句中更新之前和更新之后的行值，为了更好的可读性，还将操作列封装到了一个CASE语句中。 102

2.5.4 从CDC表查询净变更

在最初的CDC设置技巧中，为源表以@supports_net_changes = 1执行了sp_cdc_enable_table_change_data_capture。它的意思是我可以选择执行CDC过程的净变更版本。存储过程的fn_cdc_get_net_changes_version也接受了开始和结束的LSN值，但第三个参数在行筛选选项上不同。

- all，返回行最后的更改，不显示更新掩码的值。
- all with mask，返回行最后的更改和更新掩码值（下一技巧会详细说明如何解释这个掩码）。
- all with merge，返回行最终的更改，不管是删除还是合并操作（插入或更新）。插入和更新不会被打断。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

随后的技巧演示了展示净变更而不显示更新掩码。先生成一些新的数据修改：

```
INSERT dbo.Equipment
(EquipmentDESC, LocationID)
VALUES
('Portable White Board', 18)

UPDATE dbo.Equipment
SET LocationID = 1
WHERE EquipmentID = 3
```

接下来，将使用下面的查询跟踪变更的净效应：

```
DECLARE @FromLSN varbinary(10) =
  sys.fn_cdc_map_time_to_lsn
   ( 'smallest greater than or equal' , '2008-03-16 09:45:00')

DECLARE @ToLSN varbinary(10) =
  sys.fn_cdc_map_time_to_lsn
   ( 'largest less than or equal' , '2008-03-16 23:59:59')

SELECT
   CASE __$operation
      WHEN 1 THEN 'DELETE'
      WHEN 2 THEN 'INSERT'
      WHEN 3 THEN 'Before UPDATE'
      WHEN 4 THEN 'After UPDATE'
      WHEN 5 THEN 'MERGE'
   END Operation,
   __$update_mask,
   EquipmentID,
   EquipmentDESC,
   LocationID
FROM cdc.fn_cdc_get_net_changes_dbo_Equipment
   (@FromLSN, @ToLSN, 'all with mask')
```

查询返回：

Operation	__$update_mask	EquipmentID	EquipmentDESC	LocationID
INSERT	NULL	3	Portable White Board	1

103

解析

在这个技巧中，使用了cdc.fn_cdc_get_net_changes_dbo_Equipment返回行在指定的LSN范围内的净变更。首先插入一个新行，然后更新它。我查询了cdc.fn_cdc_get_net_changes_dbo_Equipment，基于LSN范围显示净变更。虽然生成了两个变更，但只需要返回一行来反映最终变更，即生成行的最终状态的INSERT操作。

2.5.5　翻译 CDC 更新掩码

通过cdc.fn_cdc_get_all_changes_和cdc.fn_cdc_get_net_changes_函数返回的更新掩码可以确定哪些列被特定操作影响了。但是为了翻译这个值，需要借助另外两个CDC函数。

- 使用sys.fn_cdc_is_bit_set检查是否在掩码中设置了指定的位。它的第一个参数是要检查的位的序号，第二个参数是更新掩码本身。
- 可以将sys.fn_cdc_get_column_ordinal和sys.fn_cdc_is_bit_set一起使用来确定表列的序号位置。这个函数的第一个参数是捕获实例的名称。第二个参数是列的名称。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在这个技巧中，将使用这两个函数来帮助识别在指定的LSN边界中更新了哪些列。首先，将对两个不同的行进行两个更新：

```
UPDATE dbo.Equipment
SET EquipmentDESC = 'HR File Cabinet A1'
WHERE EquipmentID = 2

UPDATE dbo.Equipment
SET LocationID = 35
WHERE EquipmentID = 3
```

现在将执行一个查询，使用更新掩码来确定哪些行发生了变更：

```
DECLARE @FromLSN varbinary(10) =
  sys.fn_cdc_map_time_to_lsn
   ( 'smallest greater than or equal' , '2008-03-16 10:02:00')

DECLARE @ToLSN varbinary(10) =
  sys.fn_cdc_map_time_to_lsn
   ( 'largest less than or equal' , '2008-03-16 23:59:59')

SELECT
   sys.fn_cdc_is_bit_set (
     sys.fn_cdc_get_column_ordinal (
          'dbo_Equipment' , 'EquipmentDESC' ),
               __$update_mask) EquipmentDESC_Updated,
   sys.fn_cdc_is_bit_set (
     sys.fn_cdc_get_column_ordinal (
          'dbo_Equipment' , 'LocationID' ),
               __$update_mask) LocationID_Updated,
   EquipmentID,
   EquipmentDESC,
   LocationID
FROM cdc.fn_cdc_get_all_changes_dbo_Equipment
   (@FromLSN, @ToLSN, 'all')
WHERE __$operation = 4
```

查询返回：

EquipmentDESC_Updated	LocationID_Updated	EquipmentID	EquipmentDESC	LocationID
1	0	2	HR File Cabinet A1	3
0	1	3	Portable White Board	35

解析

在这个技巧中，更新了两行。其中一个更新只对设备描述做了相应的变更，第二个更新对位置ID做了相应的变更。

为了识别是否设置了位，使用了如下函数调用：

```
SELECT sys.fn_cdc_is_bit_set (
```

这个函数调用的第一个参数是希望检查的列的序号位置。为了返回这个信息，使用了如下的函数调用：

```
sys.fn_cdc_get_column_ordinal ( 'dbo_Equipment' , 'EquipmentDESC' )
```

sys.fn_cdc_is_bit_set的第二个参数是要查明的更新掩码列名称。我在查询中引用了它和列名的别名：

```
, __$update_mask) EquipmentDESC_Updated,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

我在查询的下一行为LocationID重复了这个代码：

```
   sys.fn_cdc_is_bit_set (sys.fn_cdc_get_column_ordinal
( 'dbo_Equipment' , 'LocationID' ), __$update_mask) LocationID_Updated,
```

余下的查询都是标准的，返回变更列值和查询“所有变更”的CDC函数：

```
   DepartmentID,
   Name,
   GroupName
FROM cdc.fn_cdc_get_all_changes_dbo_Department
   (@FromLSN, @ToLSN, 'all')
```

最后，限定查询只返回类型为4的行，它是更新操作之后的行：

```
WHERE __$operation = 4
```

2.5.6　使用 LSN 边界

我将演示如何使用sys.fn_cdc_map_time_to_lsn确定最小和最大LSN边界。然而，不只是用这个函数可以定义边界。本技巧中下面的几个函数都可以用来产生LSN值。

105

- sys.fn_cdc_increment_lsn返回基于输入的LSN号的下一个LSN号。例如，你能使用这个函数将最新载入的上限边界LSN转换为下一个下限边界LSN。
- sys.fn_cdc_decrement_lsn返回输入的LSN号之前的LSN。
- sys.fn_cdc_get_max_lsn从捕获实例收集的CDC数据中返回最大的LSN。
- sys.fn_cdc_get_min_lsn从捕获实例收集的CDC数据中返回最早的LSN。

如下的示例演示了从dbo.Equipment表收集的CDC数据获取LSN值：

```
SELECT sys.fn_cdc_get_min_lsn ('dbo_Equipment') Min_LSN

SELECT sys.fn_cdc_get_max_lsn () Max_LSN

SELECT sys.fn_cdc_increment_lsn (sys.fn_cdc_get_max_lsn()) New_Lower_Bound_LSN

SELECT sys.fn_cdc_decrement_lsn (sys.fn_cdc_get_max_lsn())
New_Lower_Bound_Minus_one_LSN
```

查询返回如下信息（注意你的结果将会是不同的）：

```
Min_LSN
0x0000001C000001040014

(1行受影响)

Max_LSN
0x0000001E0000008B0001

(1行受影响)

New_Lower_Bound_LSN
0x0000001E0000008B0002

(1行受影响)

New_Lower_Bound_Minus_one_LSN
0x0000001E0000008B0000

(1行受影响)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

这个新的CDC功能提供了在数据库中跟踪目标表变更的内建的方法；但是，你必须还要考虑使用何种逻辑捕获更改跟踪的时间范围。这个技巧演示了从CDC数据库中获取最小和最大可用LSN。

sys.fn_cdc_get_min_lsn函数以捕获实例名称作为输入参数，而sys.fn_cdc_get_max_lsn返回在数据库级别最大的LSN。sys.fn_cdc_increment_lsn和sys.fn_cdc_decrement_lsn函数用来基于你传入的LSN增加和减少LSN。使用这些函数可以为CDC数据的查询创建了新的边界。

2.5.7 禁用对表和数据库的变更数据捕获

这个技巧演示了如何从表中删除变更数据捕获。为了做到这一点，将执行sys.sp_cdc_disable_table存储过程。在这个示例中，将禁用表中可能存在的所有更改跟踪：

```
EXEC sys.sp_cdc_disable_table
   'dbo', 'Equipment', 'all'
```

然后可以通过执行如下查询验证表是否确实被禁用：

```
SELECT is_tracked_by_cdc
FROM sys.tables
WHERE name = 'Equipment' and
   schema_id = SCHEMA_ID('dbo')
```

查询返回：

```
is_tracked_by_cdc
0

(1行受影响)
```

要禁用数据库本身的CDC，执行如下存储过程：

```
EXEC sys.sp_cdc_disable_db
```

查询返回：

```
命令已成功执行.
```

解析

存储过程sys.sp_cdc_disable_table用来从表中删除CDC。存储过程的第一个参数指定了架构的名称，第二个参数指定了表名。最后一个参数指定了你是否希望通过指定all或指定捕获实例的名称来删除所有更改跟踪。

执行sys.sp_cdc_disable_db过程从数据库本身完整删除CDC能力，这也删除了CDC架构与相关的SQL代理作业。

2.5.8 以最小的磁盘开销跟踪净数据更改

CDC用来对数据存储和数据仓库的持续数据变更进行异步跟踪，也提供了侦测数据中间变更的能力。不像CDC，更改跟踪是一个同步进程，是DML操作本身（INSERT/UPDATE/DELETE）事务的一部分，使用它可以以最小的磁盘存储开销来侦测净行变更。

更改跟踪的同步行为允许修改的数据以事务一致的形式表现，也具有检测数据冲突的能力。应用程序可以以最小的性能开销使用这个功能，而不必增加支持的数据库对象的修改（不需要自定义更改检测触发器或表时间戳）。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在这个技巧中，将一步步讲解如何使用新的更改跟踪功能去检测DML操作。首先，创建用来演示
107 这个功能的新数据库：

```
IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'TSQLRecipeChangeTrackDemo')
BEGIN
    CREATE DATABASE TSQLRecipeChangeTrackDemo
END
GO
```

要启用数据库的更改跟踪功能，必须配置CHANGE_TRACKING数据库选项。也可以配置在数据库中保留多久的更改，以及是否启用自动清除。配置保留期限将影响数据库维护多少更改跟踪。将这个值设得过高会影响存储。如果设得过低那么在远程应用程序同步不够的情况下，会引发与另一应用程序的同步问题：

```
ALTER DATABASE TSQLRecipeChangeTrackDemo
SET CHANGE_TRACKING = ON
(CHANGE_RETENTION = 36 HOURS,
AUTO_CLEANUP = ON)
```

使用更改跟踪时的最佳实践是为数据库启用快照隔离。对于有显著DML活动的数据库和表，以一致的方式捕获更改跟踪的信息很重要——抓取最新版本并使用该版本号来获取适当的数据。

警告　由于行版本的生成，启用快照隔离会在tempdb中增加额外的空间使用。这也增加了总的I/O开销。

不使用快照隔离会引发事务不一致的变更信息：

```
ALTER DATABASE TSQLRecipeChangeTrackDemo
SET ALLOW_SNAPSHOT_ISOLATION ON
GO
```

可以通过查询sys.change_tracking_databases来确认数据库已正确启用更改跟踪：

```
SELECT DB_NAME(database_id) DBNM,is_auto_cleanup_on,
       retention_period,retention_period_units_desc
FROM sys.change_tracking_databases
```

查询返回：

```
DBNM                        is_auto_cleanup_on   retention_period   retention_period_units_desc
TSQLRecipeChangeTrackDemo   1                    36                 HOURS
```

现在将创建一个新表，用来演示更改跟踪：

```
USE TSQLRecipeChangeTrackDemo
GO

CREATE TABLE dbo.BookStore
(BookStoreID int NOT NULL IDENTITY(1,1) PRIMARY KEY CLUSTERED,
 BookStoreNM varchar(30) NOT NULL,
 TechBookSection bit NOT NULL)
GO
```

108

下一步，对于希望跟踪更改的所有表，都要使用带有CHANGE_TRACKING选项的ALTER TABLE命令。如果还要跟踪哪些列被更新了，那么还要启用TRACK_COLUMNS_UPDATED选项，如下一步演示的：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ALTER TABLE dbo.BookStore
ENABLE CHANGE_TRACKING
WITH (TRACK_COLUMNS_UPDATED = ON)
```

通过查询sys.change_tracking_tables目录视图可以确定哪些表启用了更改跟踪：

```
SELECT OBJECT_NAME(object_id) ObjNM,is_track_columns_updated_on
FROM sys.change_tracking_tables
```

查询返回：

```
ObjNM          is_track_columns_updated_on
BookStore      1
```

现在通过用3个新行对表进行初始化填充来演示更改跟踪：

```
INSERT dbo.BookStore
(BookStoreNM, TechBookSection)
VALUES
('McGarnicles and Bailys', 1),
('Smith Book Store', 0),
('University Book Store',1)
```

用来查看正在进行的同步的一个新函数是CHANGE_TRACKING_CURRENT_VERSION函数，它返回表中最后提交的事务中的版本号。所有发生在更改跟踪表中的DML操作将造成版本号增长。稍后将使用这个版本号来确定更改：

```
SELECT CHANGE_TRACKING_CURRENT_VERSION ()
```

查询返回：

```
1
```

还可以使用CHANGE_TRACKING_MIN_VALID_VERSION函数来检查更改跟踪表的最小可用版本。如果断开连接的应用程序不同步的时间超过了更改跟踪保留期限，那么有必要对应用程序数据进行彻底的刷新。

```
SELECT CHANGE_TRACKING_MIN_VALID_VERSION
 ( OBJECT_ID('dbo.BookStore') )
```

查询返回：

```
0
```

可以使用CHANGETABLE函数来检测更改。这个函数有两种用法：使用CHANGES关键字来检测自指定的同步版本以来发生的更改，或使用VERSION关键字来返回行的最新更改跟踪版本。

109

下面首先演示CHANGES如何工作。随后的查询演示返回BookStore表自版本0以后的最近更改。第一个参数是更改跟踪表的名称，第二个参数是版本号：

```
SELECT BookStoreID,SYS_CHANGE_OPERATION,
       SYS_CHANGE_VERSION
FROM CHANGETABLE
 (CHANGES dbo.BookStore, 0) AS CT
```

它返回表的主键，随后是DML操作类型（I代表INSERT，U代表UPDATE，D代表DELETE）以及相关的行版本号（由于三行数据是在一条INSERT中添加的，因此它们共享同一个版本号）：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
BookStoreID    SYS_CHANGE_OPERATION    SYS_CHANGE_VERSION
1              I                       1
2              I                       1
3              I                       1
```

警告　当收集同步信息时，使用SET TRANSACTION ISOLATION LEVEL SNAPSHOT和BEGIN TRAN...COMMIT TRAN来封装收集的更改信息和相关的当前更改跟踪版本及最小可用版本。使用快照隔离允许更改跟踪的数据具有事务一致性的形式。

为了进一步演示更改跟踪，现在多修改几次数据：

```
UPDATE dbo.BookStore
SET BookStoreNM = 'King Book Store'
WHERE BookStoreID = 1

UPDATE dbo.BookStore
SET TechBookSection = 1
WHERE BookStoreID = 2

DELETE dbo.BookStore
WHERE BookStoreID = 3
```

我将检查最新的版本号：

```
SELECT CHANGE_TRACKING_CURRENT_VERSION ()
```

这个值增加了3（有3个操作对数据进行了操作）：

```
4
```

现在假设外部应用程序收集了数据版本1之后的数据。随后的查询演示了如何检测自版本1起发生的所有更改：

```
SELECT BookStoreID,
       SYS_CHANGE_VERSION,
       SYS_CHANGE_OPERATION,
       SYS_CHANGE_COLUMNS
FROM CHANGETABLE
 (CHANGES dbo.BookStore, 1) AS CT
```

110

它返回自版本1起被修改的行信息，显示了两个前面执行的更新的主键和删除的那行的主键：

```
BookStoreID    SYS_CHANGE_VERSION    SYS_CHANGE_OPERATION    SYS_CHANGE_COLUMNS
1              2                     U                       0x0000000002000000
2              3                     U                       0x0000000003000000
3              4                     D                       NULL
```

SYS_CHANGE_COLUMNS列是包含从最新版本开始更新过的列的varbinary值。可以使用CHANGE_TRACKING_IS_COLUMN_IN_MASK函数来解释它，正如我将要在下面演示的。这个函数接受两个参数：表列的列ID和要计算的varbinary值。随后的查询使用这个函数来检查BookStoreNM列和TechBookSection列是否被修改过：

```
SELECT BookStoreID,
 CHANGE_TRACKING_IS_COLUMN_IN_MASK(
     COLUMNPROPERTY(
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        OBJECT_ID('dbo.BookStore'),'BookStoreNM', 'ColumnId') ,
         SYS_CHANGE_COLUMNS) IsChanged_BookStoreNM,
 CHANGE_TRACKING_IS_COLUMN_IN_MASK(
      COLUMNPROPERTY(
        OBJECT_ID('dbo.BookStore'), 'TechBookSection', 'ColumnId') ,
         SYS_CHANGE_COLUMNS) IsChanged_TechBookSection
FROM CHANGETABLE
 (CHANGES dbo.BookStore, 1) AS CT
WHERE SYS_CHANGE_OPERATION = 'U'
```

它返回关于哪些列被修改的位值，1为真，0为假。

BookStoreID	IsChanged_BookStoreNM	IsChanged_TechBookSection
1	1	0
2	0	1

下一步，将演示CHANGETABLE的VERSION变量可用来返回每行最新的变更版本。为了完成更改跟踪同步，可以存储这个版本的值并通过应用程序进行跟踪：

```
SELECT bs.BookStoreID, bs.BookStoreNM, bs.TechBookSection,
       ct.SYS_CHANGE_VERSION
FROM dbo.BookStore bs
CROSS APPLY CHANGETABLE
 (VERSION dbo.BookStore, (BookStoreID), (bs.BookStoreID)) as ct
```

它返回SYS_CHANGE_VERSION列，还有每行的当前列值。

BookStoreID	BookStoreNM	TechBookSection	SYS_CHANGE_VERSION
1	King Book Store	1	2
2	Smith Book Store	1	3

现在将执行另一个UPDATE来演示版本的不同：

```
UPDATE dbo.BookStore
SET BookStoreNM = 'Kingsly Book Store',
    TechBookSection = 0
WHERE BookStoreID = 1
```

下一步，使用CHANGETABLE执行另一个查询：

```
SELECT bs.BookStoreID, bs.BookStoreNM, bs.TechBookSection,
       ct.SYS_CHANGE_VERSION
FROM dbo.BookStore bs
CROSS APPLY CHANGETABLE
 (VERSION BookStore, (BookStoreID), (bs.BookStoreID)) as ct
```

这个查询显示了我刚修改的那行的行版本，现在增加到了5——但我没有修改的那行仍然保持版本号为3：

BookStoreID	BookStoreNM	TechBookSection	SYS_CHANGE_VERSION
1	Kingsly Book Store	0	5
2	Smith Book Store	1	3

我现在将检查一下当前版本号：

```
SELECT CHANGE_TRACKING_CURRENT_VERSION ()
```

该查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
5
```

这个版本号与最后提交的事务作用在表上的最新更改相匹配。

在本技巧的最后部分，将演示如何通过DML操作提供更改跟踪应用程序上下文信息。它允许你跟踪哪一应用程序对哪些行造成了数据修改——如果对几个数据源中的数据进行同步，这将会是有用的信息。我可以使用CHANGE_TRACKING_CONTEXT函数来应用这个数据的版本谱系。这个函数只有一个输入参数context，它是varbinary数据类型的值，表示调用的应用程序。

先声明保存应用程序上下文信息的变量。然后在CHANGE_TRACKING_CONTEXT函数中使用变量，之后向更改跟踪表插入一个新行：

```
DECLARE @context varbinary(128) = CAST('Apress_XYZ' as varbinary(128));

WITH CHANGE_TRACKING_CONTEXT (@context)
INSERT dbo.BookStore
(BookStoreNM, TechBookSection)
VALUES
('Capers Book Store', 1)
```

现在，将检查从版本5开始发生的所有更改（之前使用CHANGE_TRACKING_CURRENT_VERSION获取的）：

```
SELECT BookStoreID,
       SYS_CHANGE_OPERATION,
       SYS_CHANGE_VERSION,
       CAST(SYS_CHANGE_CONTEXT as varchar) ApplicationContext
FROM CHANGETABLE
 (CHANGES dbo.BookStore, 5) AS CT
```

112　它返回插入的新行值，以及从SYS_CHANGE_CONTEXT列转换出来的应用程序上下文信息：

```
BookStoreID    SYS_CHANGE_OPERATION    SYS_CHANGE_VERSION    ApplicationContext
4              I                       6                     Apress_XYZ
```

解析

在这个技巧中，演示了如何以最小的磁盘存储开销使用更改跟踪来检测净行更改。先创建了新的数据库，然后使用ALTER DATABASE...SET CHANGE_TRACKING在数据库中启用更改跟踪。还使用CHANGE_RETENTION和AUTO_CLEANUP选项指定了一个36小时保持期的更改跟踪。使用了sys.change_tracking_databases目录视图来检查更改跟踪的数据库的状态。

本技巧中还为数据库启用了快照隔离。这是一个最佳实践，当你从更改跟踪表中检索行更改版本和相关数据时，你会希望使用快照隔离级别的事务。

下一步，创建了新表，然后使用了ALTER TABLE...ENABLE CHANGE_TRACKING。通过启用TRACK_COLUMNS_UPDATED指定了列级别更改也会被跟踪。并通过查询sys.change_tracking_tables目录视图确认了表的更改跟踪状态。

在这之后，演示了一些用来检索更改跟踪数据的不同的函数，包括：

- CHANGE_TRACKING_CURRENT_VERSION，它从表中最后提交的事务返回版本号；
- CHANGE_TRACKING_MIN_VALID_VERSION，它返回更改跟踪表的最小可用版本；
- 带有CHANGES的CHANGETABLE，检测自指定同步版本以来的更改；
- CHANGE_TRACKING_IS_COLUMN_IN_MASK，检测更改跟踪表中的哪些列被更新；

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 带有VERSION的CHANGETABLE，返回行最新的更改版本；
- CHANGE_TRACKING_CONTEXT，通过DML操作存储更改上下文，从而可以跟踪哪一应用程序修改了什么数据。

更改跟踪这一功能集可以避免为净更改跟踪方案自行编写代码。这个特性拥有最小的开销，而且不需要为了实现它而修改架构（不需要触发器或时间戳）。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第3章 事务、锁定、阻塞和死锁

前两章介绍了数据修改语言，并且提供了SELECT、INSERT、UPDATE和DELETE语句的一些技巧。在介绍数据定义语言（创建/修改/删除表、索引、视图等）之前，本章会先复习处理事务、锁监控、阻塞和死锁的一些技巧。还会复习新的SQL Server 2008表选项，使用该选项可以禁用锁升级或为分区表启用锁升级。我会演示快照隔离级别和用来监控和排除阻塞和锁定问题的动态管理视图。

3.1 事务控制

事务是关系型数据库系统不可缺少的一部分，它们用来帮助定义一个工作单元。这样的工作单元能包含一个或多个T-SQL语句，它们以一个整体进行提交或者回滚。这样的“全有全无”功能能防止部分更新或者数据状态不一致。所谓部分更新，是指当一个相关进程只有一部分被回滚或者取消，而其他部分没有回滚或回退时产生的情况。

每个事务都具有ACID校验。ACID表示原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）[或者独立性（Independence）]和持久性（Durability）。

- 原子性意味着事务是一个全有全无的实体——要么完成所有步骤要么什么都没有发生。
- 一致性确保数据在事务前和事务后都是有效的，必须保持数据完整（例如外键引用）以及内部数据结构处于一个有效的状态。
- 隔离性是事务不依赖其他并发事务（即使在同一时间或者重叠发生）的必要条件。一个事务不能看到其他处于中间状态的事务的数据，但能看到事务开始前或结束后的数据。
- 持久性意味着事务提交后事务的效果是永久的，即使系统发生故障修改也会生效。

在本章中，我会介绍和复习SQL Server的一些用于确保遵从ACID校验的机制和功能，也就是锁定和事务。

在SQL Server中有3种可能的事务类型：自动提交、显式和隐式。

自动提交（autocommit）是SQL Server默认的行为，每一个独立的T-SQL语句都会在执行完毕后自动提交。例如，如果你有两个INSERT语句，第一个失败了而第二个成功了，第二个的修改还是有效，因为每一个INSERT自动包含在自己的事务中。尽管这种模式能让开发者无需操心显式实现事务，但是对事务性的活动采用这种模式是会产生错误的。例如，你有两个事务，一个为账户还款，而另一个贷款，但是第一个事务失败了，这样你只是贷款而没有还款。遇到这种情况银行可能高兴了，但对于客户来说却未必，账户上莫明其妙多了贷款。自动提交对于特殊的管理变动有点儿危险——例如，如果你不小心删除了表中的所有行，在意识到错误后没有机会回滚事务。

当下列语句中的一个首先执行的时候，SQL Server会话将自动打开一个新事物，此时隐式（implicit）

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

事务将被启用：ALTER TABLE、FETCH、REVOKE、CREATE、GRANT、SELECT、DELETE、INSERT、TRUNCATE TABLE、DROP、OPEN和UPDATE。

只要上述任意语句被执行就会自动创建（或打开）一个新的事务，并且该事务会一直保持打开状态，直到遇到ROLLBACK或COMMIT语句。初始化命令已经包含在打开事务的过程中了。在查询会话中执行下面的命令能激活隐式模式：

```
SET IMPLICIT_TRANSACTIONS ON
```

要关闭它（回到显式模式）则执行：

```
SET IMPLICIT_TRANSACTIONS OFF
```

隐式模式在正式的应用中会很麻烦，因为应用程序设计师和最终用户可能会忘记提交事务，让它们处于打开的状态，这会阻塞其他连接（本章后面会详细介绍阻塞）。

显式（explicit）事务是由你自己定义的。当对数据库应用程序进行数据修改操作的时候，显式事务是操作最推荐的模式。这是因为你要显式控制哪些修改属于一个事务以及有错误发生的时候执行什么操作。那些需要分在一组中的修改操作按照你自己的意愿来实现。

显式事务使用下面的T-SQL命令和关键字，如表3-1所示。

表3-1 显式事务命令

命 令	描 述
BEGIN TRANSACTION	设置一个显式事务的起点
ROLLBACK TRANSACTION	恢复由一个事务修改的原始数据，使数据回到事务开始时的状态。释放由事务占据的资源
COMMIT TRANSACTION	如果没有遇到错误则结束事务并且永久实现修改。释放由事务占据的资源
BEGIN DISTRIBUTED TRANSACTION	允许我们定义一个由微软分布式事务处理协调器（MS DTC）管理的分布式事务的起点。MS DTC必须在本地并且远程运行
SAVE TRANSACTION	SAVE TRANSACTION在事务内部设置一个保存点，允许我们定义一个事务在部分取消后能返回的位置。事务回滚到保存点后必须立即回滚或提交
@@TRANCOUNT	返回连接的活动事务数量。BEGIN TRANSACTION把@@TRANCOUNT加1，而ROLLBACK TRANSACTION和COMMIT TRANSACTION把@@TRANCOUNT减1。ROLLBACK TRANSACTION到保存点不会影响@@TRANCOUNT

3.1.1 使用显式事务

这个技巧的示例会演示如何使用显式事务根据一组语句返回的错误来提交或者回滚数据修改：

```
USE AdventureWorks
GO

-- 之前的数量
SELECT COUNT(*) BeforeCount FROM HumanResources.Department

-- 用于保存最新错误整数值的变量
DECLARE @Error int

BEGIN TRANSACTION

INSERT HumanResources.Department
(Name, GroupName)
VALUES ('Accounts Payable', 'Accounting')
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SET @Error = @@ERROR
IF (@Error<> 0) GOTO Error_Handler

INSERT HumanResources.Department
(Name, GroupName)
VALUES ('Engineering', 'Research and Development')

SET @Error = @@ERROR
IF (@Error <> 0) GOTO Error_Handler

COMMIT TRAN

Error_Handler:
IF @Error <> 0
BEGIN
   ROLLBACK TRANSACTION
END

-- 之后的数量
SELECT COUNT(*) AfterCount FROM HumanResources.Department
```

这个查询返回：

```
BeforeCount
16

(1行受影响)
(1行受影响)

消息2601，级别14，状态1，第14行
不能在具有唯一索引'AK_Department_Name'的对象'HumanResources.Department'中插入重复键的行。
语句已终止。

AfterCount
16

(1行受影响)
```

117

解析

这个示例的第一个语句检测HumanResources.Department表的总行数，返回16行：

```
-- 之前的数量
SELECT COUNT(*) BeforeCount FROM HumanResources.Department
```

接着创建一个局部变量来保存@@ERROR函数的值（它捕获SQL语句的最近的错误状态）：

```
-- 用于保存最新错误整数值的变量
DECLARE @Error int
```

然后，开始了一个显式事务：

```
BEGIN TRANSACTION
```

下一个语句尝试对HumanResources.Department表执行插入操作。在部门名上有一个唯一键，但是因为在表中还不存在这样的部门名，所以插入操作成功了：

```
INSERT HumanResources.Department
(Name, GroupName)
VALUES ('Accounts Payable', 'Accounting')
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接着对INSERT进行错误处理：

```
SET @Error = @@ERROR
IF (@Error <> 0) GOTO Error_Handler
```

这行代码获取@@ERROR函数的值。@@ERROR系统函数返回当前连接范围内最后执行的语句的最后的错误代码。IF语句的意思是，如果有错误发生，代码应该跳转到代码的Error_Handler部分（使用GOTO）。

注解 要复习GOTO请参阅第9章。要复习@@Error请参阅第16章。第16章也介绍了新的错误处理命令TRY...CATCH。

GOTO是用于控制语句执行流程的关键字。GOTO后的标识符Error_Handler是一个用户定义的代码部分。

接着尝试另外一个插入操作，这次是一个表中已经存在的部门了。因为表对部门名列有唯一约束，所以插入会失败：

```
INSERT HumanResources.Department
(Name, GroupName)
VALUES ('Engineering', 'Research and Development')
```

失败会引起INSERT后的@@Error设置成一个非0值。IF语句会计算为TRUE，然后会调用GOTO，因此会跳过COMMIT TRAN直接转到Error_Handler部分：

```
SET @Error = @@ERROR
IF (@Error <> 0) GOTO Error_Handler

COMMIT TRAN
```

下面的Error_Handler部分是ROLLBACK TRANSACTION：

```
Error_Handler:
IF @Error <> 0
BEGIN
   ROLLBACK TRANSACTION
END
```

在ROLLBACK后执行了另外一个计数，数据库中又是只有16行。这是因为两个INSERT在同一个事务中，其中有一个INSERT失败了。而事务是要么全部执行要么什么都不做，因此没有行被插入：

```
-- 之后的数量
SELECT COUNT(*) AfterCount FROM HumanResources.Department
```

最后介绍一些有关如何在T-SQL代码或者应用程序中处理事务的思想和建议。

- 对要做的业务处理，事务操作的时间要尽可能短。保持事务打开会让资源在很长一段时间内处于锁定状态，这样会阻塞其他用户进行操作或读取数据。
- 最小化由事务锁定的资源。例如，只更新与要做的事务有关的表。如果数据修改在逻辑上相互依赖，则它们应该属于同一个事务，否则，不相关的更新应该在它们自己的事务中。
- 只向事务添加相关的T-SQL操作。不要向某个事务添加与它关系不大的额外的查找或更新。在事务中执行SELECT语句会在关联表上加锁，这样会导致其他用户/会话不能执行操作或读取数据。
- 不要打开需要在事务内等待用户或外部反馈的新事务。打开事务会让资源处于加锁状态，用户的反馈可能会需要不确定的一段时间来接收。因此，应该在发出显式事务之前先收集用户的反馈。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

3.1.2　使用 DBCC OPENTRAN 显示最早的活动事务

无论是不是有意的，如果事务在数据库中保持打开，则它会阻塞其他进程对修改后的数据进行操作。同样，对事务日志进行备份也只会截断不活动事务的那部分事务日志，所以打开的事务会导致日
119 志数据变多（甚至达到物理限制），直到事务被提交或者回滚。

要标识数据库中最早的活动事务，可以使用DBCC OPENTRAN命令。这个示例演示了使用DBCC OPENTRAN来标识数据库中最早的活动事务：

```
BEGIN TRANSACTION

DELETE Production.ProductProductPhoto
WHERE ProductID = 317

DBCC OPENTRAN('AdventureWorks')

ROLLBACK TRAN
```

这个查询返回：

```
(1行受影响)
数据库'AdventureWorks'的事务信息。

最早的活动事务:
    SPID (服务器进程ID): 54
    UID (用户ID): -1
    名称          : user_transaction
    LSN           : (41:1021:39)
    开始时间      : Sep 15 2008 10:45:53:780AM
    SID           : 0x0105000000000005150000000a065cf7e784b9b5fe77c8770375a2900
DBCC执行完毕。如果DBCC输出了错误信息，请与系统管理员联系。
```

解析

这个示例从打开一个新的事务开始，然后删除Production.ProductProductPhoto表的某行。然后，执行DBCC OPENTRAN，传入数据库名作为参数：

```
DBCC OPENTRAN(AdventureWorks)
```

结果显示了最早活动事务的相关信息，包括服务器进程ID、用户ID和事务的开始时间。结果中的关键信息是SPID（服务器进程ID）和Start time。

在拥有了这个信息之后，就可以使用动态管理视图来检验正在执行的T-SQL，指出进程执行了多久，以及在必要的情况下关闭这个过程。DBCC OPENTRAN对于孤立连接（在数据库中是打开的，但与应用程序或客户端已经断开的连接）的排除是有用的，并能帮助我们找出遗漏了COMMIT或ROLLBACK的事务。

这个命令也返回数据库内存在的最早的分布式和非分布式复制事务。如果没有活动事务，则不会返回任何会话级的数据。

3.1.3　通过会话查询事务信息

这个技巧演示了如何找出更多活动事务的信息。为了进行演示，将描述一个常见的场景：你的应用程序会遇到很多高持续时间的阻塞。你被告知这个应用程序经常在每个查询之前打开显式
120 的事务。

为了说明这个场景，将执行如下的SQL（表示应用程序代码引发了并发性问题）：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRAN

SELECT *
FROM HumanResources.Department

INSERT HumanResources.Department
(Name, GroupName)
VALUES ('Test', 'QA')
```

在另一个/新的SQL Server Management Studio查询窗口，通过查询sys.dm_tran_session_transactions动态管理视图（DMV）来识别所有打开的事务：

```
SELECT session_id, transaction_id, is_user_transaction, is_local
FROM sys.dm_tran_session_transactions
WHERE is_user_transaction = 1
```

这个查询返回如下内容（你实际的会话ID和事务ID会有所不同）：

```
session_id   transaction_id   is_user_transaction   is_local
54           145866           1                     1
```

现在有会话ID可以使用了，可以通过查询sys.dm_exec_connections和sys.dm_exec_sql_text来挖掘最近执行的查询的详细信息：

```
SELECT s.text
FROM sys.dm_exec_connections c
CROSS APPLY sys.dm_exec_sql_text(c.most_recent_sql_handle) s
WHERE session_id = 54
```

这个查询返回最后执行的语句（也可以为正在进行的和活动的会话使用sys.dm_exec_requests DMV，但由于我的示例现在没有执行任何内容，因此不会返回数据。）

```
text
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRAN

SELECT *
FROM HumanResources.Department

INSERT HumanResources.Department
(Name, GroupName)
VALUES ('Test', 'QA')
```

因为也从对ys.dm_tran_session_transactions的第一个查询中得到了事务的ID，所以可以使用sys.dm_tran_active_transactions来了解更多事务本身的内容：

121

```
SELECT transaction_begin_time,
      CASE transaction_type
         WHEN 1 THEN 'Read/write transaction'
         WHEN 2 THEN 'Read-only transaction'
         WHEN 3 THEN 'System transaction'
         WHEN 4 THEN 'Distributed transaction'
      END tran_type,
      CASE transaction_state
         WHEN 0 THEN 'not been completely initialized yet'
         WHEN 1 THEN 'initialized but has not started'
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        WHEN 2 THEN 'active'
        WHEN 3 THEN 'ended (read-only transaction)'
        WHEN 4 THEN 'commit initiated for distributed transaction'
        WHEN 5 THEN 'transaction prepared and waiting resolution'
        WHEN 6 THEN 'committed'
        WHEN 7 THEN 'being rolled back'
        WHEN 8 THEN 'been rolled back'
    END tran_state
FROM sys.dm_tran_active_transactions
WHERE transaction_id = 145866
```

这个查询返回了事务开始时间、事务类型以及事务状态：

```
transaction_begin_time    tran_type              tran_state
2008-08-26 10:03:26.520   Read/write transaction  active
```

解析

这个技巧演示了如何使用各种DMV来排除故障和调查长时间运行的活动事务。你决定使用哪些列取决于你要排除什么故障。在这个场景中，我使用如下的故障排除步骤：

- 查询sys.dm_tran_session_transactions来显示会话ID和事务ID（各个事务的标识符）之间的映射；
- 查询sys.dm_exec_connections和sys.dm_exec_sql_text来查找会话最新执行的命令（引用most_recent_sql_handle列）；
- 最后，查询sys.dm_tran_active_transactions来确定事务被打开了多长时间、事务的类型以及事务的状态。

使用这个故障排除技术可以回到应用程序去查明查询调用的被抛弃的事务（打开但从未提交），以及那些运行时间太长或对于应用程序来说是不必要的不恰当事务。

3.2　锁定

锁定是关系型数据库系统常规且必要的部分，它防止对相同数据作并发更新或在更新过程中查看数据，从而保证数据的完整性。锁定也能防止用户读取正在被修改的数据。SQL Server动态地管理锁定，然而还是很有必要了解T-SQL查询是怎么影响SQL Server中的锁定的。在接触技巧以前我会简单介绍一下SQL Server锁定的基本原理。

122

锁定能帮助防止并发问题的发生。当一个用户试图读取另外一个用户正在修改的数据，或者尝试修改另一个用户正在读取的数据，或者尝试修改另一个事务正在尝试修改的数据的时候，就会出现并发问题（3.3节会详细讨论）。

SQL Server资源会被锁定。资源的锁定方式称作它的锁定模式（lock mode），表3-2列出了SQL Server处理的主要锁定模式。

表3-2　SQL Server锁定模式

名　字	描　述
共享锁	共享锁由读取（非更新）查询产生。共享锁占据的时候允许其他进程读取数据但不允许修改数据
意向锁	意向锁有效地创建锁队列，指定连接的顺序和它们更新或读取资源的相关权限。SQL Server使用意向锁来标识对某资源获得锁的意向

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

名 字	描 述
更新锁	更新锁在修改数据前获得。如果行被修改，锁就被升级为排他锁；如果没有修改则被降级为共享锁。如果两个连接在一个资源上占有了一个共享（S）锁，并且都试图转化为排他（X）锁，那么它们都不能实现，因为它们都在等待对方事务释放共享（S）锁，而更新锁类型能防止此类死锁的产生
排他锁	锁住资源禁止任何类型的访问（读或写）。在INSERT、UPDATE或DELETE语句的时候产生
架构修改	执行DDL语句的时候产生
架构稳定性	在查询被编译的时候产生，使DDL操作不能在表上进行
大批量更新	在批量复制操作的时候产生这种类型的锁。批量复制操作能提升性能，但是降低了表并发性
键范围	键范围锁保护一系列的行（根据索引键）。例如，在UPDATE语句中保护日期范围是1/1/2005到12/31/2005的行。保护这个范围的数据可以防止插入到这个范围中的行被当前的数据修改弄丢

可以锁定SQL Server中的各种对象，既可以是一个数据库中的行，也可以是一个表也可以是数据库本身。可以锁定的资源在粒度上差异很大，从细（行级别）到粗（整个数据库）。细粒度锁允许更大的数据库并发，因为用户能对某些未锁定的行执行查询。然而每一个由SQL Server产生的锁都需要内存，所以数以千计独立的行级别的锁也会影响SQL Server的性能。粗粒度的锁降低了并发性，但消耗的资源也较少。表3-3详细介绍了SQL Server可以锁定的资源。

表3-3 SQL Server的可锁定资源

资源名称	描 述
Allocation unit	一系列根据数据类型分组的相关页面，例如数据行、索引行和大型对象数据行
Application	应用程序专用的资源
DB	整个数据库的锁
Extent	8个8KB数据或索引页的分配单元
File	数据库文件
HOBT	堆（没有聚集索引的表）或B树
Metadata	系统元数据
Key	用于防止幻读的索引行锁。也叫做键范围锁，这种锁类型使用范围和行组件。这个范围表示两个连续的索引键之间的索引键的范围。行组件表示索引实体上的锁类型
Object	一个数据库对象（例如视图、存储过程、函数）
Page	8KB数据或者索引页
RID	行标识符，指定一个表行
Table	锁定整个表、数据和索引的资源

不是所有的锁类型能彼此兼容。例如，一个被排他锁锁定的资源不能再加其他锁。其他事务必须等待或超时，直到排他锁被释放。被更新锁锁定的资源只能接受其他事务的共享锁。被共享锁锁定的资源还能接受其他的共享锁或更新锁。

SQL Server自动分配和升级锁。升级意味着细粒度的锁（行锁或页锁）被转化为粗粒度的表锁。当单个T-SQL语句在单个表或索引上获取5 000多个锁，或者SQL Server实例中的锁的数量超过可用内存阈值的时候，SQL Server会尝试启动锁升级。锁占用系统内存，因此把很多锁转化为一个较大的锁能释放内存资源。然而，在释放内存资源的同时会降低并发性。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

提示　SQL Server 2008带来了新的表选项，使用它可以禁用锁升级或在分区（而不是表）范围启用锁升级。3.2.2节的技巧中会演示它。

3.2.1　查看锁的活动

这个技巧介绍如何使用SQL Server sys.dm_tran_locks动态管理视图监视数据库中锁的活动。这个被DMV监视的示例查询使用数据表锁提示。

124

在技巧的第一部分，我们打开一个新的查询编辑器窗口，执行如下的命令：

```
USE AdventureWorks

BEGIN TRAN
SELECT ProductID, ModifiedDate
FROM Production.ProductDocument
WITH (TABLOCKX)
```

打开第二个查询编辑器窗口，执行下面的查询：

```
SELECT request_session_id sessionid,
      resource_type type,
      resource_database_id dbid,
      OBJECT_NAME(resource_associated_entity_id, resource_database_id) objectname,
      request_mode rmode,
      request_status rstatus
FROM sys.dm_tran_locks
WHERE resource_type IN ('DATABASE', 'OBJECT')
```

提示　为了更清晰地表现这个技巧，示例中将结果集的范围缩到了两个SQL Server资源类型：DATABASE和OBJECT。你将看到几种资源。资源类型确定resource_associated_entity_id列的含义，我会在“解析”中进行解释。

查询返回了锁会话标识符（服务器进程ID，也就是SPID）、被锁定的资源、数据库、对象、资源模式和锁状态等信息：

sessionid	type	dbid	objectname	rmode	rstatus
53	DATABASE	8	NULL	S	GRANT
52	DATABASE	8	NULL	S	GRANT
52	OBJECT	8	ProductDocument	X	GRANT

解析

在这个示例中我们首先启动了一个新的事务，并使用TABLOCKX锁提示（这个提示对表放置了排他锁）对Production.ProductDocument表执行了一个查询。查询sys.dm_tran_locks动态管理视图可以监视当前SQL Server实例中打开了哪些锁。它返回了AdventureWorks数据库中活动锁的列表。可以在结果中的最后一行看到ProductDocument表上的排他锁。

前三列定义了会话锁、资源类型和数据库ID：

```
SELECT request_session_id sessionid,
      resource_type type,
      resource_database_id dbid,
```

下一列使用了OBJECT_NAME函数。注意它使用了两个参数（对象ID和数据库ID）来指定访问哪个名称

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（第二个数据库ID参数是SQL Server 2005 SP2引入的，它用来指定为了翻译对象名称使用哪个数据库）：

```
OBJECT_NAME(resource_associated_entity_id, resource_database_id) objectname,
```

125

同时也查询锁定请求模式和状态：

```
request_mode rmode,
request_status rstatus
```

最后，FROM子句引用了DMV，然后WHERE子句指定了两个资源类型：

```
FROM sys.dm_tran_locks
WHERE resource_type IN ('DATABASE', 'OBJECT')
```

resource_type列指定了锁定的资源类型（例如DATABASE、OBJECT、FILE、PAGE、KEY、RID、EXTENT、METADATA、APPLICATION、ALLOCATION_UNIT或HOBT类型）。resource_associated_entity_id依赖资源类型，确定ID是对象ID、分配单元ID还是Hobt ID。

- 如果resource_associated_entity_id列包含对象ID（资源类型为OBJECT），可以使用sys.objects目录视图来翻译名称。
- 如果resource_associated_entity_id列包含分配单元ID（资源类型为ALLOCATION_UNIT），可以引用sys.allocation_units和container_id。然后可以将container_id联结到sys.partitions上，此时就可以确定对象ID了。
- 如果resource_associated_entity_id列包含Hobt ID（资源类型为KEY、PAGE、ROW或HOBT），可以直接引用sys.partitions，然后查找相应的对象ID。
- 对于DATABASE、EXTENT、APPLICATION或METADATA的资源类型，resource_associated_entity_id列将为0。

使用sys.dm_tran_locks能对无法预料的并发问题进行故障调试。例如，一个查询会话占用锁的时间可能比预期时间长锁，或者锁的粒度或锁模式不是我们所期望的（可能是希望使用表锁而不是更小粒度的行锁或页锁）。理解锁处于的锁定级别能帮助我们更有效地对查询的并发问题进行故障调试。

3.2.2 控制表的锁升级行为

每个在SQL Server中创建的锁都会消耗内存资源。当锁的数量增加时，内存就会减少。如果锁的内存使用百分比超过一个特定阈值，SQL Server会将细粒度锁（页或行）转换为粗粒度锁（表锁）。这个过程称为锁升级。锁升级可以减少SQL Server实例占有的锁的数量，减少锁内存的使用。

虽然细粒度锁会消耗更多的内存，但由于多个查询可以访问未锁定的行，因此也会改善并发性。引入表锁可能会减少内存的消耗，但也会带来阻塞，这是因为一条查询锁住了整个表。根据使用数据库的应用程序，这个行为可能是不希望发生的，而且你可能希望当SQL Server实施锁升级时尽量获得更多的控制。

SQL Server 2008引入了使用ALTER TABLE命令在表级别控制锁升级的功能。现在可以从如下3个设置中选择。

- TABLE，这是SQL Server 2005中使用的默认的行为。当设置为该值时，就在表级别启用了锁升级，不论是否为分区表。
- AUTO，如果表已分区，则在分区级别（堆或B树）启用锁升级。如果表未分区，锁升级将发生在表级别上。
- DISABLE在表级别删除锁升级。注意，对于用了TABLOCK提示或使用可序列化隔离级别下堆的查询时，你仍然可能看到表锁。

126

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个技巧演示了修改表的两个新的SQL Server 2008设置：

```
ALTER TABLE Person.Address
SET ( LOCK_ESCALATION = AUTO )

SELECT lock_escalation,lock_escalation_desc
FROM sys.tables
WHERE name='Address'
```

这个查询返回：

```
lock_escalation   lock_escalation_desc
2                 AUTO
```

下一步，禁用锁升级：

```
ALTER TABLE Person.Address
SET ( LOCK_ESCALATION = DISABLE)

SELECT lock_escalation,lock_escalation_desc
FROM sys.tables
WHERE name='Address'
```

这个查询返回：

```
lock_escalation   lock_escalation_desc
1                 DISABLE
```

解析

这个技巧演示了启用两个新的SQL Server 2008表选项。首先是一个标准的ALTER TABLE命令，其后指定了要修改的表名：

```
ALTER TABLE Person.Address
```

第二行指定了SET命令，以及要使用的LOCK_ESCALATION配置：

```
SET ( LOCK_ESCALATION = AUTO )
```

在更改了这个配置后，可以通过查询sys.tables目录视图的lock_escalation_desc列来验证这个选项：

注解　*更多关于分区的信息，参见第4章。*

如果表未分区，通常情况为表级别升级。如果你指定了DISABLE选项，将不会出现表级别的锁升级。
127 这会提高并发性，但如果你请求访问大量的行或页，会增加内存的消耗。

3.3　事务、锁定和并发

隔离性（isolation）是ACID中的一个属性。事务隔离是指由某个事务作出的修改能被数据库产生的其他事务看见的程度（例如在数据库访问并发的条件下）。对于最高级的隔离，每一个事务产生就好像在同一时间只有一个事务存在一样，它看不到其他事务作出的修改。对于最低级的隔离，任何事务进行的操作，无论是否提交都能被其他事务看见。

ANSI/ISO SQL标准定义了并发事务之间的4种交互类型，如下所示。

- **脏读（dirty read）**：如果一个事务正在更新一行，而第二个事务在第一个事务提交之前读取了

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

那行，就会发生这种情况。如果原来的更新回滚了，那么第二个事务读取的数据就不一样。这就是脏读。

- **不可重复读取**（norepeatable read）：当一个事务在更新数据，第二个事务在其更新过程中读取相同的数据时会发生这种情况。更新前获取的数据和更新后获取的数据不一致。
- **幻读**（phantom read）：事务产生两次读取，并且在两次读取之间底层的数据有插入的或是被删除的数据。它会造成两个查询的结果行有所不同。在一个查询中出现了但另一个查询未出现的仍称为“幻行”。
- **更新丢失**（lost update）：如果两个事务都更新了同一行的值，并且最后一次更新的事务“取得了胜利”，那么第一次的更新就丢失了。

SQL Server使用锁定机制来控制并发事务的竞争行为。为了避免脏读、不可重复读等并发问题，SQL Server通过锁来控制对数据库资源的访问，实现某种级别的事务隔离。表3-4描述了SQL Server中可用的隔离级别。

表3-4 SQL Server隔离级别

隔离级别	描 述
READ COMMITTED（这是SQL Server的默认行为）	如果设置了READ COMMITTED，就不能读取未提交的数据修改。在查询中使用共享锁，而且当查询获取数据的时候，其他进程不能修改数据。只要涉及的行没有被第一个事务锁定，其他事务就能对同一个表进行数据插入和修改
READ UNCOMMITED	这是限制最小的隔离级别，对事务检索的数据不加锁。因为读取的数据在读取时可以被修改（这些读可视为“脏读”），或者有数据新增或者删除从而改变了原始的查询结果，所以它提供了最高的并发性和最低数量的数据完整性。这个选项允许我们在不阻塞的情况下读取，但是也带来了读取不稳定数据的危险，因为数据可能在读取的过程中被修改（包括读取了一个最终被回滚的事务的数据更改）。对于相对静态或不修改的数据而言，这种隔离级别可能会提升性能，因为它指示SQL Server不要对访问的资源加不必要的锁
REPEATABLE READ	如果开启了该级别，就不允许脏读和不可重复读取。这通过为所有读取资源加共享锁来实现。然而，在查询返回的日期范围中新插入的行还是能被其他事务插入
SERIALIZABLE	如果开启了该级别，这是限制最高的设置。根据产生结果集的搜索标准，会在数据上加范围锁。这样就保证了不允许进行插入行、修改值或删除在原始查询和搜索标准中返回的行操作
SNAPSHOT	这个隔离级别允许我们读取事务一致性版本的数据，因为它存在于一个事务的开始阶段。数据读取不阻塞数据修改——然而，SNAPSHOT会话不能检测到当前的修改

事务和锁定是密切关联的。根据应用程序的设计，事务能显著地影响数据库并发。并发是指在同一时间有多少人能查询和修改数据库和数据库对象。例如，READ UNCOMMITED隔离级别不加锁，所以它允许最大数量的并发——缺点是可能会遇到一些数据孤立的异常（例如脏读）。然而，在查询大批量数据的时候，SERIALIZABLE模式提供了和其他进程最小的并发性。

配置会话的事务加锁行为

这个技巧演示如何使用SET TRANSACTION ISOLATION LEVEL命令来为连接内使用的T-SQL语句设置默认的事务加锁行为。在同一时间只可以设置一个隔离级别，除非显式设置，否则隔离级别不会改变。SET TRANSACTION ISOLATION LEVEL允许你为特定的数据库连接修改加锁行为。此命令的语法如下：

```
SET TRANSACTION ISOLATION LEVEL
    { READ UNCOMMITTED
    | READ COMMITTED
    | REPEATABLE READ
    | SNAPSHOT
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    | SERIALIZABLE
    }
```

在第一个示例中，我们使用SERIALIZABLE隔离级别来查询表中的内容。打开第一个查询编辑窗口，执行下面的代码：

```
USE AdventureWorks
GO

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
GO

BEGIN TRAN
SELECT AddressTypeID, Name
FROM Person.AddressType
WHERE AddressTypeID BETWEEN 1 AND 6
```

129

查询返回如下结果（仍然保持打开查询会话的事务）：

```
AddressTypeID   Name
1               Billing
2               Home
3               Main Office
4               Primary
5               Shipping
6               Archive
```

在第二个查询编辑器中执行下面的查询来查看由SERIALIZABLE隔离级别生成的锁类型：

```
SELECT resource_associated_entity_id, resource_type,
       request_mode, request_session_id
FROM sys.dm_tran_locks
```

查询显示request_session_id 52（这是另外一个会话的ID）占据了几个键范围锁：

```
resource_associated_entity_id  resource_type  request_mode  request_session_id
0                              DATABASE       S             52
0                              DATABASE       S             53
72057594043039744              PAGE           IS            52
101575400                      OBJECT         IS            52
72057594043039744              KEY            RangeS-S      52
72057594043039744              KEY            RangeS-S      52
72057594043039744              KEY            RangeS-S      52
72057594043039744              KEY            RangeS-S      52
72057594043039744              KEY            RangeS-S      52
72057594043039744              KEY            RangeS-S      52
72057594043039744              KEY            RangeS-S      52
```

返回第一个查询编辑器窗口，执行下面的代码来结束事务并移除锁：

```
COMMIT TRAN
```

为了进行对比，我们在第一个查询编辑器窗口中执行同样的查询，这次使用READ UNCOMMITTED隔离级别来读取某个范围的行：

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
GO

BEGIN TRAN

SELECT   AddressTypeID, Name
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FROM Person.AddressType
WHERE AddressTypeID BETWEEN 1 AND 6
```

在第二个查询编辑器中，执行下面的查询来查看READ UNCOMMITTED隔离级别生成的锁类型：

130

```
SELECT resource_associated_entity_id, resource_type,
request_mode, request_session_id
FROM sys.dm_tran_locks
```

这个查询返回（部分结果）：

```
resource_associated_entity_id   resource_type   request_mode   request_session_id
0                               DATABASE        S              52
0                               DATABASE        S              53
```

不像SERIALIZABLE隔离级别，READ UNCOMMITTED隔离级别不在Person.AddressType表的键上创建额外的锁。

返回到第一个执行READ UNCOMMITTED查询的查询编辑器，结束事务：

```
COMMIT TRAN
```

下一步将演示SNAPSHOT隔离级别。在第一个查询编辑器窗口中执行下面的代码：

```
ALTER DATABASE AdventureWorks
SET ALLOW_SNAPSHOT_ISOLATION ON
GO

USE AdventureWorks
GO

SET TRANSACTION ISOLATION LEVEL SNAPSHOT
GO

BEGIN TRAN

SELECT   CurrencyRateID,
      EndOfDayRate
FROM Sales.CurrencyRate
WHERE CurrencyRateID = 8317
```

查询返回：

```
CurrencyRateID EndOfDayRate
8317           0.6862
```

在第二个查询编辑器中执行下面的查询：

```
USE AdventureWorks
GO

UPDATE Sales.CurrencyRate
SET EndOfDayRate = 1.00
WHERE CurrencyRateID = 8317
```

返回到第一个查询编辑器，重新执行下面的查询：

```
SELECT CurrencyRateID,
   EndOfDayRate
FROM Sales.CurrencyRate
WHERE CurrencyRateID = 8317
```

131

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回：

```
CurrencyRateID   EndOfDayRate
8317             0.6862
```

即使第二个查询编辑器的查询更新了行，该查询还是返回了和前面相同的结果。SELECT没有因为读取行而被阻塞，而UPDATE也没有被阻塞进行修改。

现在将提交这个事务并重新执行这个查询：

```
COMMIT TRAN

SELECT   CurrencyRateID,
   EndOfDayRate
FROM Sales.CurrencyRate
WHERE CurrencyRateID = 8317
```

查询返回更新的信息：

```
CurrencyRateID   EndOfDayRate
8317             1.00
```

解析

在这个技巧中，我演示了如何使用SET TRANSACTION ISOLATION LEVEL来修改查询会话的锁定隔离级别。如果你希望使用SQL Server默认的隔离级别——READ COMMITTED，就没必要执行这个命令。否则，一旦你设置了隔离级别，它会保持对当前连接有效，除非显式改变。

技巧中的第一个示例演示了使用SERIALIZABLE隔离级别：

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
GO
```

然后开启了一个显式事务，并对Person.AddressType表中AddressTypeID值介于特定范围的行执行查询：

```
BEGIN TRAN

SELECT    AddressTypeID, Name
FROM Person.AddressType
WHERE AddressTypeID BETWEEN 1 AND 6
```

接着，在一个独立的连接中，对sys.dm_tran_locks动态管理视图执行查询，以返回SQL Server实例占有的活动锁的相关信息。在这个示例中，我们看到了很多键范围锁，它们用于禁止其他连接查询、更新或删除数据，避免导致查询搜索条件（WHERE AddressTypeID BETWEEN 1 AND 6）的不同结果。

在第二个示例中，把隔离级别设置成READ UNCOMMITTED：

132

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
GO
```

再一次查询sys.dm_tran_locks，我们看到这次根本没有行锁、键锁或者页面锁加在表上，这样就允许其他事务修改被查询的行，即使原始的事务处于打开状态。使用这个隔离级别，这个查询实现“脏读”，也就是说查询能在修改的过程中读取数据，无论之后真正的修改被提交还是回滚。

在这个技巧的第三个示例中，我们对数据库启用了ALLOW_SNAPSHOT_ISOLATION设置（有关ALTER DATABASE的更多信息请参阅第22章）：

```
ALTER DATABASE AdventureWorks
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SET ALLOW_SNAPSHOT_ISOLATION ON
GO
```

要开启快照事务，这个选项必须设置成ON。在代码的下一行，数据库上下文改变了，并且SET TRANSACTION ISOLATION LEVEL被设置成了SNAPSHOT：

```
USE AdventureWorks
GO

SET TRANSACTION ISOLATION LEVEL SNAPSHOT
GO
```

然后打开了一个事务，并对Sales.CurrencyRate表执行查询：

```
BEGIN TRAN
SELECT   CurrencyRateID,
      EndOfDayRate
FROM Sales.CurrencyRate
WHERE CurrencyRateID = 8317
```

在第二个查询编辑器会话中，修改了第一个会话查询选择的Sales.CurrencyRate行：

```
USE AdventureWorks
GO

UPDATE Sales.CurrencyRate
SET EndOfDayRate = 1.00
WHERE CurrencyRateID = 8317
```

回到第一个查询编辑器会话，尽管在第二个会话中把EndOfDayRate修改成了1.0，但是在SNAPSHOT隔离级别执行的查询显示了列的值仍然为0.6862。从事务开始起，这个新的隔离级别就提供一致的数据视图。

要是你打算更新一个快照会话中的行，并且那个行已经被一个独立会话更新，会怎么样呢？如果快照会话企图对CurrencyRateID为8 317的行进行UPDATE而不是SELECT，那么就会产生一个错误，警告你在快照隔离模式下对原始行进行更新：

```
消息3960，级别16，状态1，第2行
快照隔离事务由于更新冲突而中止。您无法在数据库'AdventureWorks'中使用快照隔离来直接或间接访问表
'Sales.CurrencyRate'，以便更新、删除或插入已由其他事务修改或删除的行。请重试该事务或更改update/
delete语句的隔离级别。
```

133

3.4 阻塞

当一个数据库会话中的事务正在锁定一个或多个其他会话事务想要读取或者修改的资源时，会产生阻塞。通常短时间的阻塞没有问题，且是较忙的应用程序所需要的。然而，设计得比较糟糕的应用程序会导致长时间的阻塞，这样就不必要地锁定了资源，而且阻塞了其他会话读取和更新它们。

在SQL Server中，一个阻塞的进程会无限期地保持阻塞，或者直到它超时（根据SET LOCK_TIMEOUT）、服务器关闭、进程被杀死、连接完成了更新或者其他发生在原始事务上的操作导致它释放了资源上的锁。

为什么会发生长时间的阻塞呢？有这样一些原因。

- 在一个没有索引的表上的过量的行锁会导致SQL Server得到一个表锁，从而阻塞其他事务。
- 应用程序打开一个事务，并在事务保持打开的时候要求用户进行反馈或者交互。这通常是让最终用户在GUI上输入数据而保持事务打开的时候发生。保持打开的话，事务引用的任何资源

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

都会被锁占据。

- 事务BEGIN后查询的数据可能在事务开始之前被引用。
- 查询不恰当地使用锁定提示。例如，应用程序仅使用很少的行，但是却使用了一个表锁提示（要复习锁定提示，请参阅15.1.3节介绍的可用锁定提示的列表）。
- 应用程序使用长时间运行的事务，在一个事务中更新了很多行或很多表（把一个大量更新的事务变成多个更新较少的事务能帮助改善并发性）。

3.4.1　找到并解决阻塞进程

在这个技巧中，我会演示如何找到阻塞进程，查看由该进程执行的T-SQL，然后强制终止这个活动会话的连接（因此会回滚这个阻塞会话未提交的所有工作）。不过，我们首先将快速了解一下本例中使用的命令的背景知识。

这个技巧演示如何使用SQL Server动态管理视图sys.dm_os_waiting_tasks来找出阻塞的进程。这个视图用于代替早期版本的SQL Server中使用的系统存储过程sp_who。

在找出阻塞的进程后，我们会使用本章前面使用的sys.dm_exec_sql_text动态管理函数和sys.dm_exec_connections DMV找出正在执行的查询的SQL文本——然后来强制结束进程。

要强制终止一个难以控制的活动查询会话，可以使用KILL命令。KILL应该仅用于其他方法都不可用的情况，比如等待进程自己终止或者通过调用应用程序终止或取消操作。KILL的语法如下：

```
KILL {spid | UOW} [WITH STATUSONLY]
```

134　表3-5描述了此命令的参数。

表3-5　KILL命令参数

参　数	描　述
spid	与要终止的活动数据库连接关联的会话ID
UOW	分布式事务的工作单元标识符。这是某个分布式事务进程的唯一标识符
WITH STATUSONLY	某些KILL语句回滚事务所花的时间比其他命令更长（这取决于会话中执行的更新范围）。要检查回滚的状态，可以使用WITH STATUSONLY来获取回滚的估计时间

作为示例开始，我们在第一个查询编辑器窗口中执行下面的查询来创建一个阻塞的进程：

```
BEGIN TRAN

UPDATE Production.ProductInventory
SET Quantity = 400
WHERE ProductID = 1 AND
LocationID = 1
```

然后，在第二个查询编辑器窗口中执行下面的查询：

```
BEGIN TRAN
UPDATE Production.ProductInventory
SET Quantity = 406
WHERE ProductID = 1 AND
LocationID = 1
```

现在在第三个查询编辑器窗口中执行下面的查询：

```
SELECT blocking_session_id, wait_duration_ms, session_id
FROM sys.dm_os_waiting_tasks
WHERE blocking_session_id IS NOT NULL
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回：

```
blocking_session_id   wait_duration_ms   session_id
54                    27371              55
```

这个查询找到了ID为54的会话阻塞了会话55。

要查看ID为54的会话在干什么，则在前面查询的那个窗口中执行下面的查询：

```
SELECT t.text
FROM sys.dm_exec_connections c
CROSS APPLY sys.dm_exec_sql_text (c.most_recent_sql_handle) t
WHERE c.session_id = 54
```

这个查询返回：

```
text
BEGIN TRAN
UPDATE Production.ProductInventory
SET Quantity = 400
WHERE ProductID = 1 AND
LocationID = 1
```

接着，执行下面的查询来强制终止会话：

```
KILL 54
```

查询返回：

```
命令已成功完成。
```

一旦其他会话连接被移除，就能继续进行第二个会话的UPDATE了。

解析

这个技巧演示了通过在一个打开但未提交的事务中对Production.ProductInventory表执行UPDATE来实现阻塞。我们在另一个会话中对相同的表和相同的行执行了一个相似的查询。因为另外一个连接的事务永不提交，第二个连接只能无限期排队，直到它有机会更新记录。

在第三个查询编辑器窗口中，查询了sys.dm_os_waiting_tasks动态管理视图，返回了由另一个会话阻塞的会话的信息。

在解决阻塞问题时，你可能想看到阻塞的session_id究竟在做什么。为了查看这一点，技巧中使用了对sys.dm_exec_connections和sys.dm_exec_sql_text的查询。使用sys.dm_exec_connections DMV返回了session_id为53的most_recent_sql_handle列。这是SQL文本在内存中的指针，作为sys.dm_exec_sql_text动态管理函数的输入参数使用。从sys.dm_exec_sql_text返回了text列，该列显示了阻塞进程的SQL文本。

注解 阻塞经常连接成串，必须通过blocking_session_id和session_id列仔细查看每一个阻塞的进程直到发现原始的阻塞进程。

然后使用KILL强制终止阻塞的进程，但是在正式的应用中，你会希望知道进程是否有效，如果是，能否允许完成或者能否使用应用程序来终止或取消进程（例如被应用程序的最终用户）。在停止进程前，确定不是要停止一个需要长时间执行但对业务至关重要的事务，例如一个工资单更新。如果确实没有办法停止事务（例如，产生它的应用程序不能提交事务），可以使用KILL命令（后面跟需要终止

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的SPID）来终止事务。

3.4.2　配置语句等待锁释放的时长

如果有一个事务或语句被阻塞，就意味着它在等待资源上的锁被释放。这个技巧将演示SET LOCK_TIMEOUT选项，它指定了阻塞的进程语句在返回错误之前应该等待多少时间来释放锁。

136

语法如下：

```
SET LOCK_TIMEOUT timeout_period
```

超时时段是以毫秒为单位的数字，超时后会返回锁定错误。为了进行演示，将执行下面的语句：

```
BEGIN TRAN

UPDATE Production.ProductInventory
SET Quantity = 400
WHERE ProductID = 1 AND
LocationID = 1
```

在第二个查询窗口中，该示例演示了设置超时时间为1秒（1000毫秒）：

```
SET LOCK_TIMEOUT 1000

UPDATE Production.ProductInventory
SET Quantity = 406
WHERE ProductID = 1 AND
LocationID = 1
```

在1秒钟（1000毫秒）之后，得到了如下的错误消息：

```
消息1222，级别16，状态51，第2 行
已超过了锁请求超时时段。
语句已终止。
```

解析

在这个技巧中，我们设置了锁超时时间为1 000毫秒（1秒）。这个设置不会影响资源被进程占有的时间，只会影响等待另外一个进程释放资源访问的时间。

3.5　死锁

当一个用户会话（暂且称之为会话1）已经锁定了一个资源，而另外一个用户会话（暂且称之为会话2）想要修改该资源，并且会话2也锁定了会话1想要修改的资源时，就会出现死锁（deadlocking）。在另外一方释放锁之前，会话1和会话2都不可能继续，所以SQL Server会选择死锁中的一个会话作为“死锁牺牲品”。

注解　死锁牺牲品的会话会被杀死，事务会被回滚。

下面是一些发生死锁的原因。

- 应用程序以不同的次序访问表。例如，会话1先更新了客户然后更新了订单，然而会话2先更新了订单然后是客户。与按照连续（顺序）的方式访问和更新表相比，这种方式增加了两个进程死锁的可能性。

137

- 应用程序使用了长时间运行的事务，在一个事务中更新很多行或很多表。这样就增加了行的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

“表面积”，从而导致死锁冲突。

- 在一些情况下，SQL Server发出了一些行锁，之后它又决定将其升级为表锁。如果这些行在相同的数据页面中，并且两个会话希望同时在相同的页面升级锁粒度，就会产生死锁。

3.5.1 使用跟踪标志位找出死锁

如果你的SQL Server实例有死锁问题，这个技巧会演示如何正确使用DBCC TRACEON、DBCC TRACEOFF和DBCC TRACESTATUS命令来确保死锁被正确记录到SQL Server Management Studio SQL日志中。这些命令开启、关闭和检查跟踪标志位的状态。

提示 在SQL Server中有一些其他方法来解决死锁问题，例如使用SQL Profiler；但是此书关注的是T-SQL，这些方法不在此范围内。

SQL Server使用跟踪标志位来为SQL Server实例开启或关闭某种行为。默认情况下，SQL Server在死锁事件发生时不返回重要日志。使用跟踪标志位1222，有关死锁中的被锁资源和类型的信息以XML格式返回，帮助你调试该事件。

DBCC TRACEON命令开启跟踪标志位。语法如下：

```
DBCC TRACEON ( trace# [ ,...n ][ ,-1 ] ) [ WITH NO_INFOMSGS ]
```

表3-6描述了此命令的参数。

表3-6 DBCC TRACEON命令参数

参 数	描 述
trace#	指定一个或多个需要开启的跟踪标志位数字
-1	如果指定了-1，则以全局方式打开某种（或某些）跟踪标志位
WITH NO_INFOMSGS	当命令中包含该参数时，则禁止DBCC输出信息性消息

DBCC TRACESTATUS命令用于检查某种（或某些）标志位的状态（开启或关闭）。语法如下：

```
DBCC TRACESTATUS ( [ [ trace# [ ,...n ] ] [ , ] [ -1 ] ] ) [ WITH NO_INFOMSGS ]
```

表3-7描述了此命令的参数。

表3-7 DBCC TRACESTATUS命令参数

参 数	描 述
trace#	指定一个或多个需要检查状态的跟踪标志位数字
-1	表示全局启用标志位
WITH NO_INFOMSGS	当命令中包含该参数时，则禁止DBCC输出信息性消息

DBCC TRACEOFF命令用于关闭跟踪标志位。语法如下：

```
DBCC TRACEOFF ( trace# [ ,...n ] [ , -1 ] ) [ WITH NO_INFOMSGS ]
```

表3-8描述了此命令的参数。

表3-8 DBCC TRACEOFF命令参数

参 数	描 述
trace#	指定一个或多个需要关闭的跟踪标志位数字

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

参　数	描　述
-1	关闭全局设置的标志位
WITH NO_INFOMSGS	如果在命令中包含该参数，则禁止DBCC输出信息性消息

为了演示这个技巧，我们将模拟一个死锁。在一个新的查询编辑器窗口中执行下面的查询：

```
SET NOCOUNT ON
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

WHILE 1=1
BEGIN
BEGIN TRAN

   UPDATE Purchasing.Vendor
   SET CreditRating = 1
   WHERE BusinessEntityID = 1494

   UPDATE Purchasing.Vendor
   SET CreditRating = 2
   WHERE BusinessEntityID = 1492

COMMIT TRAN
END
```

在第二个查询编辑器窗口中执行下面的查询：

```
SET NOCOUNT ON
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

WHILE 1=1
BEGIN
BEGIN TRAN

   UPDATE Purchasing.Vendor
   SET CreditRating = 2
   WHERE BusinessEntityID = 1492

   UPDATE Purchasing.Vendor
   SET CreditRating = 1
WHERE BusinessEntityID = 1494

COMMIT TRAN
END
```


几秒后，检查每一个查询编辑器窗口，直到在其中一个查询编辑器窗口中出现下面的错误消息：

```
消息1205，级别13，状态51，第9行
事务(进程ID 53)与另一个进程被死锁在锁资源上，并且已被选作死锁牺牲品。请重新运行该事务。
```

查看SQL Server Management Studio的SQL日志，发现死锁事件没有被记录。

现在打开第三个查询编辑器窗口，执行下面的命令：

```
DBCC TRACEON (1222, -1)
GO
DBCC TRACESTATUS
```

DBCC TRACESTATUS显示了本地和全局会话中活动的跟踪：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
TraceFlag   Status   Global   Session
1222        1        1        0
```

为了模拟另外一个死锁，将重启动“胜利”的那个连接查询（没有在死锁中被杀死的那个查询），然后重启动死锁丢失的会话，在几秒之后又出现另外一个死锁了。

死锁发生后，停止另外一个执行的查询。现在，SQL Server Management Studio的SQL日志中包含了死锁事件的详细错误消息，包括相关的数据库和对象、锁定模式以及死锁中的T-SQL语句。

例如，当死锁发生时，你会希望能找出死锁中的查询，从而进行故障调试。下面的日志片段显示了一个死锁查询：

```
09/15/2008 20:20:00,spid15s,Unknown,
UPDATE [Purchasing].[Vendor] set [CreditRating] = @1
WHERE [BusinessEntityID]=@2
```

从这里我们可以知道哪个查询包含在死锁中，哪个很频繁，需要解决。使用跟踪1222还能获取一些其他的重要信息，比如死锁进程的登录名、用来提交查询的客户端应用程序以及为连接使用的隔离级别（让你知道那个连接是否正在使用一个不允许很多并发性的隔离级别）：

```
... clientapp=Microsoft SQL Server Management Studio - Query hostname=CAESAR
hostpid=2388 loginname=CAESAR\Administrator isolationlevel=serializable (4)
xactid=1147351 currentdb=8 lockTimeout=4294967295
clientoption1=673187936 clientoption2=390200
```

在检查SQL日志后，在查询编辑器中关闭跟踪标志位：

```
DBCC TRACEOFF (1222, -1)
GO
DBCC TRACESTATUS
```

解析

在这个技巧中，我使用了两个独立的查询来模拟一个死锁，这两个查询重复更新同样的行：以相反的次序更新两行。当死锁发生时，错误消息记录到了查询编辑器窗口，但是SQL日志中没有写入任何东西。

为了让死锁记录到SQL日志中，我们在技巧中启用了跟踪标志位1222。跟踪1222能把详细的死锁信息返回到SQL日志中。标志位-1表明跟踪标志位1222应该对所有SQL Server连接在全局启用。要打开跟踪标志位，就要使用DBCC TRACEON，在圆括号中包含1222标志位：

```
DBCC TRACEON (1222, -1)
```

要验证标志位是否已经启动，则执行DBCC TRACESTATUS：

```
DBCC TRACESTATUS
```

之后再遇到死锁，死锁信息就会记录到SQL日志中。然后可以使用DBCC TRACEOFF来关闭标志位：

```
DBCC TRACEOFF (1222, -1)
```

3.5.2 设置死锁优先级

可以使用SET DEADLOCK_PRIORITY命令来增加一个查询会话被选为死锁牺牲品的可能性。此命令的语法如下：

```
SET DEADLOCK_PRIORITY { LOW | NORMAL | HIGH | <numeric-priority> }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表3-9描述了此命令的参数。

表3-9　SET DEADLOCK_PRIORITY命令参数

参　数	描　述
LOW	LOW使得当前的连接很可能成为死锁牺牲品
NORMAL	NORMAL使SQL Server根据哪个连接的回滚代价最小来决定
HIGH	HIGN减少连接被选为牺牲品的可能性。除非另外一个连接也是HIGH或者数值优先级大于5
<numeric-priority>	数值优先级允许你使用-10～10的值。-10最可能成为死锁牺牲品，10最不可能被选为牺牲品。两个参与者谁数字大谁就能在死锁中获胜

例如，如果前面技巧中的第一个查询使用了下面的死锁优先级命令，则它几乎肯定会被选为死锁牺牲品（正常情况下，SQL Server会把它认为取消或回滚代价最小的连接作为默认的死锁牺牲品）：

```
SET NOCOUNT ON
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
SET DEADLOCK_PRIORITY LOW

BEGIN TRAN
   UPDATE Purchasing.Vendor
   SET CreditRating = 1
   WHERE BusinessEntityID = 2

   UPDATE Purchasing.Vendor
   SET CreditRating = 2
   WHERE BusinessEntityID = 1

COMMIT TRAN
```

141

解析

也可以将死锁优先级设置为HIGH或NORMAL。HIGH表示除非另外一个会话有相同的优先级，否则它不会被选为牺牲品。NORMAL是默认行为，如果另外一个会话是HIGH，则它可能会被选中，如果另外一个会话是LOW则它不会被选中。如果两个会话都有相同的优先级，则回滚代价最小的事务会被选中。

142

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第4章 表



在本章中，我会介绍一些演示表的创建和操作的技巧。表用于在数据库中保存数据，它是大多数SQL Server数据库对象所依赖的核心单位。表在数据库和架构中唯一命名，并且包含一列或多列。每一列都有关联的数据类型，定义能保存在列里的数据种类。

提示 SQL Server 2008包含了新的数据类型，用来处理日期和时间、层次结构、空间（geography与geometry）和FILESTREAM属性。我将在第14章讨论层次结构和空间数据类型，而在本章讨论FILESTREAM属性。

和前面章节一样，我会从头到尾提供一些基本的表技巧，并穿插一些比较复杂的功能。关于SQL Server 2008引入的新特性，我将在4.1.5节的技巧中演示稀疏列的改进。4.7节中也将介绍可在SQL Server 2008的企业版或开发版中使用的新数据压缩功能。

警告 如果你决定遵循一些技巧做一些练习，可以考虑先备份AdventureWorks数据库，这样结束后你就能恢复到一个干净的版本。

4.1 表基础

可以使用CREATE TABLE命令创建表。完整的语法非常长，所以本章会根据章节的进度介绍命令不同的区域。简化的语法如下：

```
CREATE TABLE
    [ database_name . [ schema_name ] . | schema_name . ] table_name
        ( column_name <data_type> [ NULL | NOT NULL ] [ ,...n ] )
```

表4-1描述了此命令的参数。

表4-1 CREATE TABLE参数

参数	描述
database_name .[schema_name].\| schema_name .] table_name	这个参数表示你可以使用数据库名、架构名和表名或仅仅是架构名和表名来限定新的表名
column_name	列的名字
data_type	列的数据类型（数据类型稍后会描述）
NULL \| NOT NULL	NULL \| NOT NULL选项指列的可空性。可空性定义了列是否可以包含NULL值。NULL代表值未知，而不代表列是0、空白或者空

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

每一列都需要定义一个数据类型。数据类型定义并限制了列能保存的数据类型。

表4-2详细介绍了SQL Server中可用的系统数据类型：

表4-2　SQL Server数据类型

数据类型	值　范　围
bigint	这个数据类型指定从–2^63 (–9 223 372 036 854 775 808) 到 2^63–1 (9 223 372 036 854 775 807)的整数
binary	这个数据类型指定固定长度的二进制数据，最多8 000字节
bit	这个数据类型指定了整数0或1
char	这个数据类型指定了固定长度的字符数据，最大长度8 000字符
date	这个数据类型以1天为精度存储日期，范围是从1-01-01到9999-12-31
datetime	这个数据类型提供了以0.333秒为精度存储的日期和时间，范围从1753年1月1日到9999年12月31日（1753年是采用公历的那年，它和早先的历法在日期上相差12天。从1753年开始回避了各种计算问题。）
datetime2	这个数据类型以100纳秒的精度存储日期和时间，范围是从1-01-01 00:00:00.0000000到 9999-12-31 23:59:59.9999999
datetimeoffset	时区偏移量是指指定时区与协调世界时（UTC）之间的时间差。可以将这个新数据类型和SYSDATETIMEOFFSET系统函数一起使用来存储当前系统时间戳与数据库时区。也可以将这个数据类型与SWITCHOFFSET函数一起使用，通过指定某个增加量或减少量来更改时区偏移量。这个数据类型以100纳秒的精度存储，范围从1-01-01 00:00:00.0000000到9999-12-31 23:59:59.9999999
decimal或numeric（两个类型一样）	这个数据类型的范围是从–10^38 +1到10^38–1。decimal有精度和小数位数限制。精度确定最多可以存储的十进制数字的总位数，包括小数点左边和右边的位数。小数位数确定小数点右边最多可以存储的十进制数字的位数
float	这个数据类型指定了一个浮点数，范围为–1.79E + 38至–2.23E – 38、0以及2.23E – 38至1.79E + 38
geography和geometry	这两个数据类型指定空间数据的本地存储。geometry数据类型表示平面球（欧几里得）坐标系中的空间数据，也可以存储点、多边形、曲线和集合。geography数据类型用于圆球空间存储，允许存储纬度和经度坐标，也可以存储点、多边形、曲线和集合。这些数据类型是SQL Server 2008新引入的，会在第14章中讨论
hierarchyid	这个数据类型本地存储树结构的位置。这个类型是SQL Server 2008中新引入的，会在第14章中讨论
int	这个数据类型指定从2^31 (–2 147 483 648) 到 2^31–1 (2 147 483 647)的一个整数
money	这个数据类型指定从 – 2^63 (–922 377 203 685 477.580 8) 到 2^63–1 (+922 337 203 685 477.580 7)的一个货币值
nchar	这个数据类型指定了最大长度为4 000字符的定长Unicode字符数据
nvarchar	这个数据类型指定了最大长度为4 000个字符的变长Unicode字符数据。SQL Server 2005也增加了一个max选项，允许你存储最多2^31–1字节的数据。有了这个新的选项我们可以使用常规数据类型来替代SQL Server 2000的text、ntext和image
real	这个数据类型指定了一个浮点数，范围为–3.40E + 38至–1.18E – 38、0以及1.18E – 38至3.40E + 38
smalldatetime	这个数据类型表示从1900年1月1日到2079年6月6日的日期和时间
smallint	这个数据类型指定了从–32 768到32 767的整数
smallmoney	这个数据类型指定了从–214 748.364 8到214 748.364 7的货币值

144

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

数据类型	值 范 围
sql_variant	这个数据类型能保存除text、ntext、timestamp、varchar(max)、nvarchar(max)、varbinary(max)、xml、image、用户定义类型以及另外一个sql_variant类型之外的所有数据类型
table	table数据类型不能在CREATE TABLE的时候作为列类型。它用于表变量或者表值函数中行的存储
time	这个数据类型存储了精度为100纳秒的时间，范围从00:00:00.0000000 到23:59:59.9999999
timestamp	这个数据类型指定了数据库级别的唯一数字，当行修改后会更新
tinyint	这个数据类型指定了从0到255的整数
uniqueidentifier	这个数据类型存储一个16字节的GUID（全球唯一标识符）
varbinary	这个数据类型指定了最大长度为8 000字节的变长数据。SQL Server 2005也增加了一个max值，允许你存储最多2^31-1字节的数据。这个新的选项能让我们使用常规数据类型来替代SQL Server 2000的text、ntext和image
varchar	这个数据类型指定了最大长度为8 000字节的变长字符数据。SQL Server 2005也增加了一个max值，允许你存储最多2^31-1字节的数据。这个新的选项能让我们使用常规数据类型来替代SQL Server 2000的text、ntext和image
xml	这个数据类型保存本地的XML数据

145

下面是一些为列选择数据类型的基本的指导原则。

- 在字符类型列（char、nchar、vchar、nvarchar）中保存字符数据，在数值类型列（int、bigint、tinyint、smallmoney、money、decimal/numeric、float）中保存数字数据，并在smalldate、date、datetime2、time、datetimeoffset或datetime数据类型列中保存日期和/或时间数据。例如，尽管你可以在基于字符的字段中保存数值和datetime信息，但如果你尝试在数学或者其他T-SQL函数中使用这些列值的话，这样做可能会降低性能。
- 如果你的字符数据类型的列一直使用相同或者相似长度的字符，建议使用固定长度的数据类型（char、nchar）。固定长度的列为每行分配固定长度的存储空间，无论它们是否被完全利用。反之，如果你认为字符类型的列每行的长度差异会很大，则使用变长数据类型（varchar、nvarchar）。变长列会增加一些存储空间开销，但是它们只为使用的字符分配存储空间。如果你确定字符串会有固定长度，并且大部分字符串的值会被显示，那么只需要使用char或者nchar。
- 选择保存数据需要的最小的数值或者字符数据类型来。你可能喜欢选取比你需要的存储空间多的数据类型，这样会导致存储空间的浪费。减少列的空间（特别是对于非常大的表）能增加8KB的数据页中能保存的行数，减少数据库需要的总存储空间，并且可能提升索引性能（更小的索引键）。

一个表最多能有1 024列（在SQL Server 2008 RTM版中的稀疏列是例外），但是每行实际使用的字节总数不能超过8 060。一个数据页大小为8KB，其中包括存储了该页信息的头。大值数据类型varchar(max)、nvarchar(max)、varbinary(max)、text、image或xml则不受这个字节限制的约束。

另外一个不遵守8 060字节限制规则的是SQL Server为普通varchar、nvarchar、varbinary和sql_variant数据类型引入的行溢出功能。如果这些数据类型的长度都没有超过8 000字节，但是几列加起来超过8 060字节的行限制，则最大宽度的那个列会被动态地移到另外一个8KB的页，并且在原始表中

146

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使用24字节的指针代替。行溢出为管理很大的行提供了额外的灵活性，但是可能的话你还是应该在定义表的时候限制最大可变数据类型的长度，由于查询的时候需要获取更多数据页面，依赖页面溢出可能会降低查询性能。

4.1.1 创建表

在这个技巧中，我们创建一个由Person架构拥有、名称为EducationType的简单表：

```
USE AdventureWorks
GO

CREATE TABLE Person.EducationType
   (EducationTypeID int NOT NULL,
   EducationTypeNM varchar(40) NOT NULL)
```

解析

在这个示例中，我们在AdventureWorks数据库中使用Person架构创建了一个非常简单的两列的表。代码的第一行指定了架构和表名：

```
CREATE TABLE Person.EducationType
```

列的定义在第二行的圆括号内：

```
(EducationTypeID int NOT NULL,
   EducationTypeNM varchar(40) NOT NULL)
```

第一列名为EducationTypeID，它定义为一个整数类型并且指定了NOT NULL（也就是说该列不允许NULL值）。第二列名为EducationTypeNM，数据类型是varchar(40)且拥有NOT NULL选项。

在下一技巧中，你会学到如何为一个既有表添加列。

4.1.2 为既有表添加列

在表创建后，可以使用ALTER TABLE命令来修改它。像CREATE TABLE一样，本章会在基于任务的部分演示ALTER TABLE和CREATE TABLE功能。在这个技巧中，我会演示如何为既有表添加一列。

添加一列的语法如下：

```
ALTER TABLE table_name
ADD { column_name data_type } NULL
```

表4-3描述了此命令的参数。

表4-3 ALTER TABLE ADD列参数

参 数	描 述
table_name	要添加列的表名
column_name	列的名字
data_type	列的数据类型

这个示例演示向既有表中添加一列（注意，使用这个方法是在表定义最后列的位置添加列）：

```
ALTER TABLE HumanResources.Employee
ADD Latest_EducationTypeID int NULL
```

147

解析

ALTER TABLE用于对既有表作修改。代码的第一行指定了需要添加列的表：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ALTER TABLE HumanResources.Employee
```

代码的第二行定义了新的列和数据类型：

```
ADD Latest_EducationTypeID int NULL
```

当向一个已有数据的表中添加列时，你就需要添加一个允许NULL值的列。你不能指定列为NOT NULL，因为你没有办法在为表添加列时的同时为新列赋值。默认情况下，表中每行的新列的值为NULL。

4.1.3 修改列定义

除了向表中添加新列，还可以使用ALTER TABLE来修改既有列的定义。

实现这个操作的语法如下：

```
ALTER TABLE table_name
ALTER COLUMN column_name
[type_name] [NULL | NOT NULL] [COLLATE collation_name]
```

表4-4详细介绍了此命令的参数。

表4-4 ALTER TABLE...ALTER COLUMN参数

参 数	描 述
table_name	包含需要修改的列的表名
column_name	需要修改的列名
type_name	需要修改的列的数据类型
NULL \| NOT NULL	需要修改的可空性选项
COLLATE collation_name	需要修改的列排序规则（对于基于字符的数据类型）。排序规则定义了3个设置：用于保存非Unicode字符数据类型的代码页、非Unicode字符数据类型的排列次序以及Unicode数据类型的排列次序。排序规则会在4.2节介绍

这个示例演示如何改变既有表列的可空性和数据类型。HumanResources.Employee表中的Gender列最初是NOT NULL，并且LoginID列的初始数据类型是nvarchar(256)：

```
-- 让它可空
ALTER TABLE HumanResources.Employee
ALTER COLUMN Gender nchar(1) NULL

-- 将nvarchar(256)扩展为nvarchar(300)
ALTER TABLE HumanResources.Employee
ALTER COLUMN LoginID nvarchar(300) NOT NULL
```

解析

在这个技巧中，我们修改了HumanResources.Employee表的两列。使用ALTER COLUMN修改Gender列为允许NULL值，但还是保留原来的数据类型：

```
ALTER COLUMN Gender nchar(1) NULL
```

在第二个ALTER TABLE中，我们把LoginID列的数据类型从nvarchar(256)扩展为nvarchar(300)：

```
ALTER COLUMN LoginID nvarchar(300) NOT NULL
```

这种修改列的方法有一些局限性。例如，除非列数据类型是varchar、nvarchar或varbinary，否则不能修改用在索引中的列——即使那样，数据类型的新大小必须比原来的大小大。也不能对有主键或者外键约束的列使用ALTER COLUMN。对于其他列修改限制（有很多）的完整列表，可以参阅SQL Server联机丛书。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

4.1.4　创建计算列

在CREATE TABLE或ALTER TABLE语句中定义的列可以源于独立的或者基于列的计算。如果需要在相关查询中对相同的数据重复计算，则计算列就变得很有用。计算列以创建表或修改表的时候定义的表达式为基础，除非使用了PERSISTED关键字，否则计算列不会物理保存在表中。

在这个技巧中，我会给出创建计算列的演示，并且介绍使用SQL Server 2005的PERSISTED选项的方法。通过CREATE或ALTER TABLE增加一个计算列的语法如下：

```
column_name AS computed_column_expression
[ PERSISTED ]
```

column_name是新列的名字。computed_column_expression是你为了得到列的值而执行的计算。增加PERSISTED关键字能让计算的结果被物理保存。

在这个示例中，我们向既有表中加了一个新的计算列：

```
ALTER TABLE Production.TransactionHistory
ADD CostPerUnit AS (ActualCost/Quantity)
```

前面的示例创建了一个名为CostPerUnit的计算列。下一个查询利用它返回数量超过10的最高的CostPerUnit：

```
SELECT TOP 1 CostPerUnit, Quantity, ActualCost
FROM Production.TransactionHistory
WHERE Quantity > 10
ORDER BY ActualCost DESC
```

这个查询返回：

```
CostPerUnit    Quantity    ActualCost
132.0408       13          1716.5304
```

149

下一个示例创建一个PERSISTED计算列，这意味着计算后的数据实际上会物理地保存在数据库中（但仍然是由SQL Server自动计算）：

```
CREATE TABLE HumanResources.CompanyStatistic
(CompanyID int NOT NULL,
 StockTicker char(4) NOT NULL,
 SharesOutstanding int NOT NULL,
 Shareholders int NOT NULL,
 AvgSharesPerShareholder AS (SharesOutStanding/Shareholders) PERSISTED)
```

解析

第一个示例向Production.TransactionHistory表添加了一个名为CostPerUnit的非持久化列，使它在SELECT查询中能像普通表列那样被引用：

```
ADD CostPerUnit AS (ActualCost/Quantity)
```

计算列不能用DEFAULT或者FOREIGN KEY约束。计算列不能被显式更新或插入（因为它的值都是计算出的）。

计算列能用在索引中，但是一定要符合一些条件。比如是确定的（对于一组给定的输入总是返回相同的结果）和精确的（不包含浮点值）。

第二个示例演示了在CREATE TABLE命令中使用计算列：

```
AvgSharesPerShareholder AS (SharesOutStanding/Shareholders) PERSISTED
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

和第一个示例不同，增加了PERSISTED意味着数据被物理地保存在数据库中。对计算中使用的列进行的任何修改都会引起存储值的再一次更新。但是保存的数据仍然不可以直接修改——数据仍然要经过计算。然而和非持久化的方式不同，保存数据意味着列能用于表分区（请参阅4.7.1节的“实现表分区”技巧）或者非精确（基于浮点）值的索引。

4.1.5 减少 NULL 列的存储空间

SQL Server 2008引入了稀疏列，它是一种优化的存储方式，为NULL值启用零字节的存储。因此，可以为表定义大量的稀疏列（到编写本书时，可允许30 000个稀疏列）。当数据库设计和应用程序需要大量不常填充的列，或表中列集只和表中存储数据的子集相关时，这个改进是理想的。

要定义一个稀疏列，只需要在CREATE或ALTER TABLE命令中的列定义之后添加SPARSE存储属性即可，如下面的查询演示的：

```
CREATE TABLE dbo.WebsiteProduct
(WebsiteProductID int NOT NULL PRIMARY KEY IDENTITY(1,1),
 ProductNM varchar(255) NOT NULL,
 PublisherNM varchar(255) SPARSE NULL,
 ArtistNM varchar(150) SPARSE NULL,
 ISBNNBR varchar(30) SPARSE NULL,
 DiscsNBR int SPARSE NULL,
 MusicLabelNM varchar(255) SPARSE NULL)
```

前面的表以不太规范的方式创建了只应用到指定产品类型的列。例如，PublisherNM和ISBNNBR列应用于书籍产品，而DiscsNBR、ArtistNM和MusicLabelNM更常用于音乐产品。当存储一个产品行时，没有应用的稀疏列将不会为NULL值分配存储空间。

150

继续这个演示，插入两个新行到表中（一条为书籍，另一条为音乐专辑）：

```
INSERT dbo.WebsiteProduct
(ProductNM, PublisherNM, ISBNNBR)
VALUES
('SQL Server 2008 Transact-SQL Recipes',
 'Apress',
 '1590599802')

INSERT dbo.WebsiteProduct
(ProductNM, ArtistNM, DiscsNBR, MusicLabelNM)
VALUES
('Etiquette',
 'Casiotone for the Painfully Alone',
 1,
 'Tomlab')
```

查询稀疏列的方法有多种。下面是使用标准查询方法的示例：

```
SELECT ProductNM, PublisherNM,ISBNNBR
  FROM dbo.WebsiteProduct
  WHERE ISBNNBR IS NOT NULL
```

这个查询返回：

```
ProductNM                          PublisherNM   ISBNNBR
SQL Server 2008 Transact-SQL Recipes  Apress     1590599802
```

第二个方法是使用列集。使用列集可以对所有定义在表中的稀疏列进行逻辑分组。xml数据类型计算列允许SELECT和数据修改，它通过在列定义之后指定COLUMN_SET FOR ALL_SPARSE_COLUMNS进行定

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

义。一个表只可以有一个列集，并且也不可以在已经定义了稀疏列的表增加列集。在下一个示例中，我将重新创建前面的表，不过这次会包含列集：

```
DROP TABLE dbo.WebsiteProduct

CREATE TABLE dbo.WebsiteProduct
(WebsiteProductID int NOT NULL PRIMARY KEY IDENTITY(1,1),
 ProductNM varchar(255) NOT NULL,
 PublisherNM varchar(255) SPARSE NULL,
 ArtistNM varchar(150) SPARSE NULL,
 ISBNNBR varchar(30) SPARSE NULL,
 DiscsNBR int SPARSE NULL,
 MusicLabelNM varchar(255) SPARSE NULL,
 ProductAttributeCS xml COLUMN_SET FOR ALL_SPARSE_COLUMNS)

-- 重新插入数据
INSERT dbo.WebsiteProduct
(ProductNM, PublisherNM, ISBNNBR)
VALUES
('SQL Server 2008 Transact-SQL Recipes',
 'Apress',
 '1590599802')
INSERT dbo.WebsiteProduct
(ProductNM, ArtistNM, DiscsNBR, MusicLabelNM)
VALUES
('Etiquette',
 'Casiotone for the Painfully Alone',
 1,
 'Tomlab')
```

151

定义了列集之后，就可以引用它，而不是引用稀疏列：

```
SELECT ProductNM, ProductAttributeCS
  FROM dbo.WebsiteProduct
  WHERE ProductNM IS NOT NULL
```

这个查询返回：

```
ProductNM                          ProductAttributeCS
SQL Server 2008 Transact-SQL Recipes  <PublisherNM>Apress</PublisherNM><ISBNNBR>
                                   1590599802</ISBNNBR>
Etiquette                          <ArtistNM>Casiotone for the Painfully Alone
                                   </ArtistNM><DiscsNBR>1</DiscsNBR>< MusicLabelNM>
                                   Tomlab</ MusicLabelNM>
```

正如在前面的结果中看到的，每一行都展示了显示所有非NULL列值元素的无类型的XML数据。可以使用INSERT和UPDATE来修改所有稀疏列的值。随后的查询演示了添加一个新行：

```
INSERT dbo.WebsiteProduct
  (ProductNM,ProductAttributeCS)
  VALUES
   ('Roots & Echoes',
    '<ArtistNM>The Coral</ArtistNM>
     <DiscsNBR>1</DiscsNBR>
     <MusicLabelNM>Deltasonic</ MusicLabelNM>')
```

任何未在DML操作中引用的稀疏列都将被置为NULL值。一旦表定义了列集，执行SELECT *查询将不会再返回所有稀疏列，如下面的查询演示的（只有非稀疏列和列集）：

```
SELECT *
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FROM dbo.WebsiteProduct
```

WebsiteProductID	ProductNM	ProductAttributeCS
1	SQL Server 2008 Transact-SQL Recipes	<PublisherNM>Apress</PublisherNM><ISBNNBR>1590599802</ISBNNBR>
2	Etiquette	<ArtistNM>Casiotone for the Painfully Alone</ArtistNM><DiscsNBR>1</DiscsNBR><MusicLabelNBR>Tomlab</MusicLabelNBR>
3	Roots & Echoes	<ArtistNM>The Coral</ArtistNM><DiscsNBR>1</DiscsNBR><MusicLabelNM>Deltasonic</MusicLabelNNM>

152

但是，你仍然可以显式地命名每个你希望查看的稀疏列，而不是查看整个稀疏列：

```
SELECT ProductNM, ArtistNM
FROM dbo.WebsiteProduct
WHERE ArtistNM IS NOT NULL
```

这个查询返回：

ProductNM	ArtistNM
Etiquette	Casiotone for the Painfully Alone
Roots & Echoes	The Coral

解析

使用稀疏列存储属性，可以在一个表中存储30 000个不常填充的列。如这个技巧中所演示的，定义SPARSE列和在列定义中增加名称一样容易：

```
CREATE TABLE dbo.WebsiteProduct
...
ArtistNM varchar(150) SPARSE NULL,
...
```

稀疏列可以使用很多数据类型，但image、ntext、text、timestamp、geometry、geography或用户定义数据类型除外。

警告 稀疏列也比常规非稀疏、非空列在非空值上增加了更多必要的空间。

这个技巧也演示了列集的使用，列集在CREATE TABLE的列定义中定义（在没有其他列集或稀疏列的情况下，也可以使用ALTER TABLE来增加）：

```
CREATE TABLE dbo.WebsiteProduct
...
ProductAttributeCS xml COLUMN_SET FOR ALL_SPARSE_COLUMNS)
```

当表中包含成千上万的稀疏列时，列集特别有用，因为它可以避免你直接在你的查询中引用稀疏列的名称。列集允许查询和DML操作。当实施INSERT或UPDATE时，所有未引用的稀疏列都被置为NULL，并且拥有零字节的存储空间。

4.1.6 删除表中的列

可以使用ALTER TABLE从既有表中删除列。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

实现这个操作的语法如下：

```
ALTER TABLE table_name
DROP COLUMN column_name
```

153 表4-5描述了此命令的参数。

表4-5 ALTER TABLE...DROP COLUMN参数

参 数	描 述
table_name	包含需要删除的列的表名
column_name	要从表中删除的列的名字

这个技巧演示了如何从既有表中删除列：

```
ALTER TABLE HumanResources.Employee
DROP COLUMN Latest_EducationTypeID
```

解析

代码的第一行指定了要删除列所在的表：

```
ALTER TABLE HumanResources.Employee
```

第二行指定了要从表中删除的列（包括任何存储在其中的数据）：

```
DROP COLUMN Latest_EducationTypeID
```

仅当列没有使用PRIMARY KEY、FOREIGN KEY、UNIQUE或CHECK CONSTRAINT（本章会介绍这些约束类型）时，才可以删除列。也不能删除用在索引中或者绑定了DEFAULT值的列。

4.1.7 报告表信息

系统存储过程sp_help返回有关某个表的信息，包括列定义、IDENTITY列、ROWGUIDCOL、文件组位置、索引（和键）、CHECK、DEFAULT、FOREIGN KEY约束以及关联的视图等。

这个系统存储过程的语法如下：

```
sp_help [ [ @objname = ] ' name ' ]
```

这个示例演示如何报告对象或表的详细信息（返回结果这里就不显示了，因为它们包含好几列和多个结果集）：

```
EXEC sp_help 'HumanResources.Employee'
```

解析

系统存储过程sp_help返回几个不同的结果集，包含有关某个对象的有用信息（在这个示例中，它返回HumanResources.Employee表的数据）。这个系统存储过程还能用于收集其他数据库对象的信息。

4.1.8 删除表

154 在这个技巧中，我会演示如何删除表。DROP命令使用如下的语法：

```
DROP TABLE schema.tablename
```

DROP TABLE接受一个参数，即表名。在这个示例中，删除了Person.EducationType表：

```
DROP TABLE Person.EducationType
```

解析

DROP命令从数据库中永久移除表定义和它的数据。在这个示例中，如果另外一个表在外键约束

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

中引用了这个表的主键，则DROP命令会失败。如果有外键约束引用，必须在删除主键表之前先删除它们。

4.2 排序规则基础

如果数据库需要国际化或多语言的数据存储，默认SQL Server实例的设置可能就不能满足任务的要求。这个技巧介绍如何使用排序规则查看和操作代码页以及设置排列顺序。SQL Server排序规则决定数据如何排序、比较、表现和存储。

SQL Server允许两种类型的排序规则：Windows或SQL。Windows排序规则是SQL Server的首选，因为它们提供了更多选项并且和微软Windows环境提供的支持一致。SQL排序规则用于早期的SQL Server版本，并且提供了向后兼容性。

除了SQL Server和数据库级别的排序规则设置，各个列也可以有自己的排序规则设置。如果你需要在某个列中保存字符数据，而该列使用的默认排序规则与数据库或服务器级别的排序规则不同，则可以在列定义中使用COLLATE命令。

对于varchar、char、nchar、nvarchar数据类型的列，可以在CREATE TABLE或ALTER TABLE操作中显式定义Windows或者SQL排序规则。

排序规则定义了3个设置：

- 用于保存非Unicode字符数据类型的代码页；
- 非Unicode字符数据类型的排列顺序；
- Unicode数据类型的排列顺序。

SQL Server实例的默认排序规则在安装时候定义，你可以使用默认的排序规则，也可以显式修改它。接下来的两个技巧会演示如何查看SQL Server实例中的排序规则信息和为表中的列显式定义排序规则。

4.2.1 查看排序规则元数据

可以使用SERVERPROPERTY函数和Collation选项来确定SQL Server实例的默认设置。例如：

```
SELECT SERVERPROPERTY('Collation')
```

这个查询返回（对于本例的SQL Server实例）：

```
SQL_Latin1_General_CP1_CI_AS
```

除了SQL Server实例的默认排序规则设置，也可以对数据库定义一个默认排序规则。可以使用DATABASEPROPERTYEX系统函数来确定数据库默认的排序规则。例如，下一个查询确定AdventureWorks数据库默认的数据库排序规则（第一个参数是数据库名，第二个是要查看的Collation选项）：

```
SELECT DATABASEPROPERTYEX ( 'AdventureWorks' , 'Collation' )
```

这个查询返回下面数据库的排序规则信息（在本例中它和SQL Server实例默认的一样，除非去显式修改它）：

```
SQL_Latin1_General_CP1_CI_AS
```

注解 有关SERVERPROPERTY和DATABASEPROPERTYEX函数的更多信息请参阅第8章。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

但是这些排序规则函数的结果是什么意思呢？要确定真正应用到SQL Server实例或者数据库上的排序规则设置，你可以查询fn_helpcollations表函数来获取更友好的信息。在本例中，返回了SQL_Latin1_General_CP1_CI_AS排序规则的描述：

```
SELECT description
FROM sys.fn_helpcollations()
WHERE name = 'SQL_Latin1_General_CP1_CI_AS'
```

这个查询返回排序规则的描述：

```
description
Latin1-General, case-insensitive, accent-sensitive, kanatype-insensitive, widthinsensitive
for Unicode Data, SQL Server Sort Order 52 on Code Page 1252 for nonUnicode Data
```

这个结果显示了排序规则代码页的一些细节条目，比如是否区分大小写、排序和Unicode选项等。

解析

这个技巧演示了如何查看SQL Server实例和某个数据库的默认排序规则。我们也学习了如何使用fn_helpcollations列出排序规则的代码页、是否区分大小写、排序和Unicode选项。当你得知当前的数据库环境在使用怎样的配置后，如果需要国际化的话你或许可以为表列应用不同的排序规则。在下一个技巧中会演示这些。

4.2.2　指定列的排序规则

在这个技巧中，我会演示如何使用ALTER TABLE命令为表的列指定排序规则：

```
ALTER TABLE Production.Product
ADD IcelandicProductName nvarchar(50) COLLATE Icelandic_CI_AI,
UkrainianProductName nvarchar(50) COLLATE Ukrainian_CI_AS
```

156

解析

在这个技巧中，我们向Production.Product表中添加了两个新的列。查询从使用ALTER TABLE和表名开始：

```
ALTER TABLE Production.Product
```

然后使用了ADD，后跟新的列名、数据类型、COLLATE关键字和排序规则名（对于排序规则名的列表，可以使用前面说的fn_helpcollations函数来得到）：

```
ADD IcelandicProductName nvarchar(50) COLLATE Icelandic_CI_AI,
UkrainianProductName nvarchar(50) COLLATE Ukrainian_CI_AS
```

要知道，如果你为同一个SQL Server实例中的同一个数据库或者多个数据库定义不同的排序规则，有的时候会遇到兼容性问题。跨排序规则的联结可能会不正确，数据传输也会发生数据丢失或者数据错误的情况。

4.3　键

主键（primary key）是一种特殊的约束，它标识一个或者一组列，它们依次唯一标识表中的所有行。

约束（constraint）是对要进入列中的数据进行限制。主键强制实体完整性（entity integrity），也就是说行必须是明确的和唯一的。数据库规范化的最佳实践是为每一个表创建主键。主键提供了访问记录的方式，并且确保键是唯一的。主键列不能包含NULL值。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

一个表只能有一个主键，当指定主键后，表索引（index）会自动创建，默认为聚集索引（有关索引类型可以参阅第5章）。如果你对一个表只允许有一个聚集索引有更好的用处，你也可以在主键创建的时候显式指定创建一个非聚集索引。在主键上创建的索引也记入表允许的索引总数。

要为一个列指定主键，在列定义中使用下面的语法：

```
( column_name <data_type> [ NULL | NOT NULL ] PRIMARY KEY )
```

PRIMARY KEY关键字包含在列定义的最后。

组合主键（compositive primary key）是表中唯一超过一列的组合。要定义组合主键，必须使用**表约束**（table constraint）而不是**列约束**（column constraint）。在列定义中把一个列设置为主键称作列约束。在列定义之外定义主键（单一或组合）称为表约束。

用于定义主键的表约束语法如下：

```
 CONSTRAINT constraint_name PRIMARY KEY
(column [ ASC | DESC ] [ ,...n ] )
```

表4-6详细介绍了此命令的参数。

157

表4-6 表约束、主键参数

参 数	描 述
constraint_name	这个参数指定了要添加的约束的唯一名
column [ASC \| DESC] [,...n]	组成主键的一列或者多列必须能唯一标识表中的一行（对所有指定的列，任何两行能没有相同的值）。ASC（升序）和DESC（降序）选项定义了列在聚集或非聚集索引中的排列次序

外键约束（foreign key constraint）在表之间发起和强制一些关系来帮助保持参照完整性，这意味着每一个外键列中的值也必须在被参照表对应的列中存在。外键约束也有助于定义域的完整性，它们定义了某一个列或者多个列可能的或者允许的值域。域的完整性定义了列中值的有效性。

外键约束的基本语法如下：

```
CONSTRAINT constraint_name
FOREIGN KEY (column_name)
REFERENCES [ schema_name.] referenced_table_name [ ( ref_column ) ]
```

表4-7详细介绍了此命令的参数。

表4-7 外键约束参数

参 数	描 述
constraint_name	外键约束名
column_name	当前表中引用主键表的主键列的列
[schema_name.] referenced_table_name	包含要被当前表引用的主键的表名
ref_column	要引用的主键列

接下来的一些技巧会以实际例子来演示主键和外键。

4.3.1 创建有主键的表

在这个技巧中，我会创建一个有单列主键的表：

```
CREATE TABLE Person.CreditRating(
   CreditRatingID int NOT NULL PRIMARY KEY,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    CreditRatingNM varchar(40) NOT NULL)
GO
```

在前面的示例中，主键定义在单个列上。当然，你也可以创建组合主键。

158 在这个示例中，我们创建一个新的表，它有一个由两列构成的PRIMARY KEY表约束：

```
CREATE TABLE Person.EmployeeEducationType (
    EmployeeID int NOT NULL,
    EducationTypeID int NOT NULL,
    CONSTRAINT PK_EmployeeEducationType
    PRIMARY KEY (EmployeeID, EducationTypeID))
```

解析

在这个技巧的第一个示例中，创建了一个具有单列主键的Person.CreditRating表。在列定义的后面有PRIMARY KEY关键字：

```
CreditRatingID int NOT NULL PRIMARY KEY,
```

主键列定义在列级别，而第二个示例是在表级别定义主键列：

```
CONSTRAINT PK_EmployeeEducationType
    PRIMARY KEY (EmployeeID, EducationTypeID))
```

约束的定义在列定义之后。在约束的名字之后跟随约束的类型（PRIMARY KEY），然后圆括号中是构成主键的列。

4.3.2 为既有表增加主键约束

在这个技巧中，会演示如何使用ALTER TABLE和ADD CONSTRAINT给既有表加上主键：

```
CREATE TABLE Person.EducationType
    (EducationTypeID int NOT NULL,
    EducationTypeNM varchar(40) NOT NULL)

ALTER TABLE Person.EducationType
ADD CONSTRAINT PK_EducationType
PRIMARY KEY (EducationTypeID)
```

解析

在这个技巧中，我们使用ALTER TABLE给一个还没有定义主键的既有表加上了新的主键。代码的第一行定义了需要加入主键的表名：

```
ALTER TABLE Person.EducationType
```

代码的第二行定义了约束名：

```
ADD CONSTRAINT PK_EducationType
```

前面示例中代码的最后一行声明了约束类型PRIMARY KEY，在之后的圆括号中是定义这个键的列：

159

```
PRIMARY KEY (EducationTypeID)
```

4.3.3 创建一个有外键引用的表

在这个技巧中，会演示如何创建一个有外键的表。在这个示例中，我们在CREATE TABLE语句的定义中定义了两个外键引用：

```
CREATE TABLE Person.EmployeeCreditRating(
    EmployeeCreditRating int NOT NULL PRIMARY KEY,
    BusinessEntityID int NOT NULL,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    CreditRatingID int NOT NULL,
    CONSTRAINT FK_EmployeeCreditRating_Employee
    FOREIGN KEY(BusinessEntityID)
    REFERENCES HumanResources.Employee(BusinessEntityID),
    CONSTRAINT FK_EmployeeCreditRating_CreditRating
    FOREIGN KEY(CreditRatingID)
    REFERENCES Person.CreditRating(CreditRatingID)
)
```

解析

在这个示例中，创建了一个具有两个外键引用的表。代码的前4行定义了表名和它的3个列：

```
CREATE TABLE Person.EmployeeCreditRating(
    EmployeeCreditRating int NOT NULL PRIMARY KEY,
    BusinessEntityID int NOT NULL,
    CreditRatingID int NOT NULL,
```

接下来的一行定义了第一个外键约束名（必须是当前数据库中的唯一名）：

```
CONSTRAINT FK_EmployeeCreditRating_Employee
```

之后定义了约束类型，后跟表的列（它会引用外部的主键表）：

```
FOREIGN KEY(BusinessEntityID)
```

定义引用的表，并在圆括号中定义表的主键列：

```
REFERENCES HumanResources.Employee(BusinessEntityID),
```

然后为CreditRatingID列创建第二个外键，它引用Person.CreditRating表的主键：

```
    CONSTRAINT FK_EmployeeCreditRating_CreditRating
    FOREIGN KEY(CreditRatingID)
    REFERENCES Person.CreditRating(CreditRatingID)
)
```

就如在本例中演示的那样，一个表能有多个外键——并且每一个外键能基于单一键或者引用超过一列（引用组合主键或者唯一索引）的多重（组合）键。而且，尽管外键引用和主键的列名不需要一致，但是主键/唯一列必须要有一致的数据类型。另外，你不能在跨数据库或者服务器的表上定义外键约束。 160

4.3.4 为既有表增加外键

使用ALTER TABLE和ADD CONSTRAINT能为既有表增加外键。这个操作的语法如下：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
FOREIGN KEY (column_name)
REFERENCES [ schema_name.] referenced_table_name [ ( ref_column ) ]
```

表4-8详细介绍了此命令的参数。

表4-8 ALTER TABLE...ADD CONSTRAINT参数

参 数	描 述
table_name	接收新的外键约束的表名
constraint_name	外键约束名
column_name	当前表中引用主键表的主键列的列
[schema_name.] referenced_table_name	包含要被当前表引用的主键的表名
ref_column	要引用的主键列

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个示例为既有表加上外键约束：

```
CREATE TABLE Person.EmergencyContact (
   EmergencyContactID int NOT NULL PRIMARY KEY,
   BusinessEntityID int NOT NULL,
   ContactFirstNM varchar(50) NOT NULL,
   ContactLastNM varchar(50) NOT NULL,
   ContactPhoneNBR varchar(25) NOT NULL)

ALTER TABLE Person.EmergencyContact
ADD CONSTRAINT FK_EmergencyContact_Employee
FOREIGN KEY (BusinessEntityID)
REFERENCES HumanResources.Employee (BusinessEntityID)
```

解析

这个示例演示了为既有表增加外键约束。代码的第一行定义了要加外键的表：

```
ALTER TABLE Person.EmergencyContact
```

第二行定义了约束名：

```
ADD CONSTRAINT FK_EmergencyContact_Employee
```

第三行定义了表中的列，它会引用主键表的主键：

161

```
FOREIGN KEY (BusinessEntityID)
```

最后一行代码定义了主键表和主键列的名字：

```
REFERENCES HumanResources.Employee (BusinessEntityID)
```

4.3.5 创建递归外键引用

表中的外键列能被定义成引用它自己的主键/唯一键。这个技术通常用于表现递归关系，在接下来的示例中我会演示。在这个示例中，创建了一个引用它自己的主键作为外键的表：

```
CREATE TABLE HumanResources.Company
   (CompanyID int NOT NULL PRIMARY KEY,
    ParentCompanyID int NULL,
    CompanyName varchar(25) NOT NULL,
    CONSTRAINT FK_Company_Company
    FOREIGN KEY (ParentCompanyID)
    REFERENCES HumanResources.Company(CompanyID))
```

在表中加入一个指定CompanyID和CompanyName的行：

```
INSERT HumanResources.Company
(CompanyID, CompanyName)
VALUES(1, 'MegaCorp')
```

添加第二行，这次引用的ParentCompanyID和前面插入的行相等：

```
INSERT HumanResources.Company
(CompanyID, ParentCompanyID, CompanyName)
VALUES(2, 1, 'Medi-Corp')
```

尝试插入第三行，这次为表中不存在的CompanyID指定ParentCompanyID：

```
INSERT HumanResources.Company
(CompanyID, ParentCompanyID, CompanyName)
VALUES(3, 8, 'Tiny-Corp')
```

该查询返回了如下的错误消息：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
消息547，级别16，状态0，第1行
INSERT语句与FOREIGN KEY SAME TABLE约束"FK_Company_Company"冲突。
该冲突发生于数据库"AdventureWorks"，表"HumanResources.Company",列'CompanyID'。
语句已终止。
```

解析

在这个示例中，创建了HumanResources.Company表，并定义了CompanyID列为其主键，且在ParentCompanyID上定义了引用CompanyID的外键：

```
CONSTRAINT FK_Company_Company
    FOREIGN KEY (ParentCompanyID)
    REFERENCES HumanResources.Company(CompanyID)
```


要处理父子层次，外键列ParentCompanyID必须是可空的。一个父值是NULL的公司位于公司层次的顶部（也就是说它没有父公司）。在表创建后，插入了3个新行。

第一行插入的公司没有指定ParentCompanyID（也就是说这个公司的ParentCompanyID列的值是NULL）：

```
INSERT HumanResources.Company
(CompanyID, CompanyName)
VALUES(1, 'MegaCorp')
```

第二行插入创建的公司引用了前面INSERT语句中定义的第一个MegaCorp公司。“1”这个值对于ParentCompanyID列是有效的，因为它是指前面插入的行：

```
INSERT HumanResources.Company
(CompanyID, ParentCompanyID, CompanyName)
VALUES(2, 1, 'Medi-Corp')
```

第三行尝试创建一个新的公司，ParentCompanyID是8，而它在表中不存在：

```
INSERT HumanResources.Company
(CompanyID, ParentCompanyID, CompanyName)
VALUES(3, 8, 'Tiny-Corp')
```

由于CompanyID为8的公司在表中不存在，外键约束阻止行插入并报告了一个错误。行没有被插入。

4.3.6 允许外键的级联修改

外键限制了外键列中能放入什么值。如果相关的主键或唯一值在引用表中不存在，那么对表行的INSERT或UPDATE会失败。这种限制是双向的，如果尝试删除一个主键而外键表中的一行引用了这个键，也会返回错误。因此，在删除目标主键或唯一值之前，需要先删除所有引用外键的行，否则会出现错误。

SQL Server提供了一种自动机制来处理主键/唯一键列中的修改，称作级联修改（cascading change）。在前面的技巧中，我们没有使用级联选项。可以使用ON DELETE和ON UPDATE为删除或修改操作开启级联修改。级联选项的基本语法如下：

```
[ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
[ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
[ NOT FOR REPLICATION ]
```

表4-9详细描述了此命令的参数。

表4-9 级联修改参数

参 数	描 述
NO ACTION	新外键的默认设置是NO ACTION，也就是说如果尝试删除的行的主键/唯一列的值被一个外键表引用，会得到一个错误并且语句不会执行

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

参　数	描　述
CASCADE	对于ON DELETE，如果选择了CASCADE，引用已删除主键的外键行也会被删除。对于ON UPDATE，引用已更新主键的外键行也会被更新
SET NULL	如果主键行被删除，外键引用的行会被设置成NULL（假设外键列允许NULL值）
SET DEFAULT	如果主键行被删除，外键引用的行会被设置成DEFAULT值。新的级联SET DEFAULT选项假设列设置为默认值，如果没有默认值但允许空的话就被设置成NULL
NOT FOR REPLICATION	NOT FOR REPLICATION选项用于防止外键约束被SQL Server复制代理进程处理（允许数据通过不按照主键表数据的次序复制产生）

163

在这个示例中，我们使用级联选项创建表：

```
--删除表的旧版本
DROP TABLE Person.EmployeeEducationType

CREATE TABLE Person.EmployeeEducationType(
   EmployeeEducationTypeID int NOT NULL PRIMARY KEY,
   BusinessEntityID int NOT NULL,
   EducationTypeID int NULL,
   CONSTRAINT FK_EmployeeEducationType_Employee
   FOREIGN KEY(BusinessEntityID)
   REFERENCES HumanResources.Employee(BusinessEntityID)
   ON DELETE CASCADE,
   CONSTRAINT FK_EmployeeEducationType_EducationType
   FOREIGN KEY(EducationTypeID)
   REFERENCES Person.EducationType(EducationTypeID)
   ON UPDATE SET NULL)
```

解析

在这个技巧中，在CREATE TABLE定义中为一个外键约束使用了ON DELETE CASCADE：

```
CONSTRAINT FK_EmployeeEducationType_Employee
   FOREIGN KEY(BusinessEntityID)
   REFERENCES HumanResources.Employee(BusinessEntityID)
   ON DELETE CASCADE
```

使用这个级联选项，如果HumanResources.Employee表中的某行被删除，则Person.EmployeeEducationType表中引用EmployeeID的所有行也会被删除。

在CREATE TABLE中定义的第二个外键约束使用了ON UPDATE：

```
CONSTRAINT FK_EmployeeEducationType_EducationType
   FOREIGN KEY(EducationTypeID)
   REFERENCES Person.EducationType(EducationTypeID)
   ON UPDATE SET NULL
```

如果Person.EducationType表的主键有了修改，则引用的Person.EmployeeEducationType表中的EducationTypeID列会被设置成NULL。

164

4.4　代理键

代理键（surrogate key）也叫人造键（artificial key），可以用作主键，而且没有任何内在的业务/数据意义。代理键和数据独立，用于为表提供唯一的记录定位。代理主键的一个巨大优势在于它们不需要修改。如果使用业务数据来定义键（自然键），比如名和姓，随着时间的推移它们的值可能会不断修改。代理键不需要修改，因为它们只在表的上下文之内有意义。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下面几个技巧会演示使用IDENTITY属性列和uniqueidentifier数据类型列生成和管理代理键的方法。

使用IDENTITY列属性，可以为表中的一个列定义一个自动增长的数字值。IDENTITY列是代理主键列中最常用的，因为它们比非数字数据类型的自然键更紧凑。当一个新行插入到有IDENTITY列属性的表中的时候，该列会插入一个唯一的递增值。IDENTITY列的数据类型可以是int、tinyint、smallint、bigint、decimal或者numeric。一个表只能定义一个标识列，并且定义的IDENTITY列不能有DEFAULT值或与它相关的规则设置。

IDENTITY属性列的基本语法如下：

```
[ IDENTITY [ ( seed ,increment ) ] [NOT FOR REPLICATION] ]
```

IDENTITY属性接受两个值：种子和增量。种子（seed）定义了IDENTITY列的起始数字，而增量（increment）定义了为了得到表中的下一行的值而加到前面的IDENTITY列的值。默认的种子和增量都是1。NOT FOR REPLICATION选项能在复制的时候保存IDENTITY列数据的原始值，保留被外键约束引用的任何值（防止使用IDENTITY列作为主键和外键引用的那些表之间的关系被打破）。

和IDENTITY列不同，它确保定义的表中数据的唯一性，而ROWGUIDCOL属性确保一个很高级别的唯一（微软称全球每一个联网的数据库都会是唯一的）。这对于那些从多个数据源合并数据的应用程序来说非常重要，唯一值不能跨表复制。这个唯一ID存储在uniqueidentifier数据类型中，由NEWID系统函数生成。ROWGUIDCOL是列定义中指定的记号，允许我们不仅仅根据列的名，还能根据ROWGUIDCOL来查询，这个技巧会演示。

哪种代理键数据类型是首选的呢？尽管使用uniqueidentifier数据类型和NEWID值作为主键更唯一，但是它比基于整数值的IDENTITY列占用的空间更多。如果你只关心值在表中的唯一性，或许应该使用整数的代理键，特别是对于非常大的表来说。然而，如果唯一性是必需的，你也可能在将来合并数据源，那么uniqueidentifier和NEWID或许是你的最佳选择。

接下来的一组技巧会以实际的例子演示IDENTITY和ROWGUIDCOL属性。

4.4.1 在创建表的时候使用 IDENTITY 属性

在这个示例中，我会演示如何创建一个将IDENTITY列作为主键的表。IDENTITY关键字放在可空选项之后，但在PRIMARY KEY关键字之前：

```
CREATE TABLE HumanResources.CompanyAuditHistory
   (CompanyAuditHistory int NOT NULL IDENTITY(1,1) PRIMARY KEY,
    CompanyID int NOT NULL ,
    AuditReasonDESC varchar(50) NOT NULL,
    AuditDT datetime NOT NULL DEFAULT GETDATE())
```

165

在新表中插入两行：

```
INSERT HumanResources.CompanyAuditHistory
(CompanyID, AuditReasonDESC, AuditDT)
VALUES
(1, 'Bad 1099 numbers.', '6/1/2009')

INSERT HumanResources.CompanyAuditHistory
(CompanyID, AuditReasonDESC, AuditDT)
VALUES
(1, 'Missing financial statement.', '7/1/2009')
```

尽管两次插入都没有显式填充CompanyAuditHistory列，但是查询表后发现，列上的IDENTITY属性使得值被填充了：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SELECT CompanyAuditHistory, AuditReasonDESC
FROM HumanResources.CompanyAuditHistory
```

这个查询返回：

```
CompanyAuditHistory AuditReasonDESC
1                   Bad 1099 numbers.
2                   Missing financial statement.
```

解析

在这个示例中，我们为新表定义了IDENTITY列。IDENTITY属性在列定义之后且PRIMARY KEY定义前指定：

```
CompanyAuditHistory int NOT NULL IDENTITY(1,1) PRIMARY KEY
```

创建好表后，向表中插入了两行，但没有显式插入CompanyAuditHistory列的值。在查询表后，可以看到这两行都根据IDENTITY属性被自动分配了值，由种子的值1开始，每个新行增加1。

4.4.2　使用 DBCC CHECKIDENT 来查看和纠正 IDENTITY 种子值

在这个技巧中，我会介绍如何使用DBCC CHECKIDENT命令来为表检查列的IDENTITY值。DBCC CHECKIDENT检查某表当前最大的值。这个命令的语法如下：

```
DBCC CHECKIDENT
( 'table_name' [ , {NORESEED | { RESEED [ , new_reseed_value ] }}])
[ WITH NO_INFOMSGS ]
```

166　表4-10详细描述了此命令的参数。

表4-10　CHECKIDENT参数

参　数	描　述
table_name	这个参数表示要对其IDENTITY值进行检查的表名
NORESEED \| RESEED	NORESEED代表仅仅是汇报当前最大的标识值。RESEED指定当前的IDENTITY应该为什么值
new_reseed_value	这个参数指定新的当前的IDENTITY值
WITH NO_INFOMSGS	如果在命令中包含WITH NO_INFOMSGS，则它禁止DBCC输出信息性消息

在这个示例中，我们检查表当前的IDENTITY值：

```
DBCC CHECKIDENT('HumanResources.CompanyAuditHistory', NORESEED)
```

这个查询返回：

```
检查标识信息：当前标识值'2'，当前列值'2'。
DBCC执行完毕。如果DBCC输出了错误信息，请与系统管理员联系。
```

第二个示例把种子值重设为一个更大的数字：

```
DBCC CHECKIDENT ('HumanResources.CompanyAuditHistory', RESEED, 50)
```

这个查询返回：

```
检查标识信息：当前标识值'2'，当前列值'50'。
DBCC执行完毕。如果DBCC输出了错误信息，请与系统管理员联系。
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

第一个示例演示了使用DBCC CHECKIDENT和NO RESEED选项检查当前的IDENTITY值。第二个示例演示了真正把IDENTITY值设置成更大的值。将来的任何插入都会从这个值开始。

为什么作这样的修改呢？带RESEED的DBCC CHECKIDENT通常用于填补主键的缺口。如果你从表中删除了一些行，这些行的IDENTITY列具有最大值，那么用过的标识值不会在下次记录插入到表中的时候重用。例如，如果上次插入的行值是22，而且你删除了那行，那么接下去插入的行值会是23。那是因为值删除之后SQL Server不会填补这个空隙。如果你希望重用键值（通常在数据库的测试阶段可以这样做，但是在正式环境下你千万别重用主键值），可以在大量行删除后使用DBCC CHECKIDENT来重用数字。

4.4.3 使用 ROWGUIDCOL 属性

首先创建一个在列数据类型定义之后，在默认值定义（通过NEWID系统函数填充）之前使用ROWGUIDCOL的表：

```
CREATE TABLE HumanResources.BuildingAccess
  (  BuildingEntryExitID uniqueidentifier ROWGUIDCOL DEFAULT NEWID(),
   EmployeeID int NOT NULL,
   AccessTime datetime NOT NULL,
   DoorID int NOT NULL)
```

然后向表中插入一行：

```
INSERT HumanResources.BuildingAccess
(EmployeeID, AccessTime, DoorID)
VALUES (32, GETDATE(), 2)
```

然后，使用ROWGUIDCOL而不是原始的BuildingEntryExitID列名来查询表（原始列名也可以使用——但ROWGUIDCOL提供了一个更通用的方法来查询标识符）：

```
SELECT   ROWGUIDCOL,
      EmployeeID,
      AccessTime,
      DoorID
FROM   HumanResources.BuildingAccess
```

这个查询返回：

BuildingEntryExitID	EmployeeID	AccessTime	DoorID
92ED29C7-6CE4-479B-8E47-30F6D7B2AD4F	32	2008-09-15 16:45:04.553	2

解析

这个技巧从创建一个具有uniqueidentifier数据类型列的表开始：

```
BuildingEntryExitID uniqueidentifier ROWGUIDCOL DEFAULT NEWID(),
```

列绑定了NEWID函数作为默认值——该函数返回一个uniqueidentifier数据类型的值。除此之外，还分配了ROWGUIDCOL属性。一个表只能定义一个ROWGUIDCOL列，但还是能在表中有多个uniqueidentifier列。

之后的SELECT查询使用ROWGUIDCOL来返回uniqueidentifier列，尽管它可以使用列名来替代。

4.5 约束

SQL Server使用约束来强制列数据的完整性。主键和外键都是约束的一种形式。其他用于列的约

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

束形式包括：

- UNIQUE约束，它强制表的非主键列的唯一性；
- DEFAULT约束，当将某行首次插入表中但不知道某列的值时可以使用它，但是仍然希望使用期望的值来填充列；
- CHECK约束，用于定义列允许的数据格式和允许的值。

168

下面的几个技巧会演示如何创建和管理这些约束类型。

4.5.1　创建唯一约束

一个表只能定义一个主键。如果希望为其他非主键列强制唯一性，可以使用UNIQUE约束。根据定义，唯一约束创建一个替代键。和PRIMARY KEY约束不同，你可以为一个表创建多个UNIQUE约束，并且也可以为允许NULL值的列指定UNIQUE约束（每个表的单列键只允许一个NULL值）。和主键相似的是，UNIQUE约束通过确保行被唯一标识来保证实体完整性。

UNIQUE约束在创建的时候会创建一个基础表索引。这个索引可以是CLUSTERED的或者NONCLUSTERED的，但是在表已经存在聚集索引的时候不能创建CLUSTERED索引。

和PRIMARY KEY约束一样，可以在创建表的时候在列定义或者表约束级别定义一个UNIQUE约束。在创建表的时候定义一个UNIQUE约束的语法如下：

```
( column_name <data_type> [ NULL | NOT NULL ] UNIQUE )
```

这个示例演示了创建一个定义了PRIMARY KEY和UNIQUE键的表：

```
CREATE TABLE HumanResources.EmployeeAnnualReview(
   EmployeeAnnualReviewID int NOT NULL PRIMARY KEY,
   EmployeeID int NOT NULL,
   AnnualReviewSummaryDESC varchar(900) NOT NULL UNIQUE)
```

可以通过创建表约束来对多个列应用唯一约束：

```
 CONSTRAINT constraint_name UNIQUE
(column [ ASC | DESC ] [ ,...n ] )
```

表4-11详细描述了此命令的参数。

表4-11　UNIQUE约束参数

参　数	描　述
constraint_name	这个参数指定了要增加的约束的唯一名
column [ASC \| DESC] [,...n]	存储在列中的值必须唯一标识表中的一行（比如，对于所有指定列，任意两行都不可以有相同的值）。ASC（升序）和DESC（降序）选项定义了列在聚集和非聚集索引中的排列顺序

在这个示例中，根据3个表列创建了一个具有UNIQUE约束的新表：

```
-- 删除表的旧版本
DROP TABLE Person.EmergencyContact

CREATE TABLE Person.EmergencyContact (
   EmergencyContactID int NOT NULL PRIMARY KEY,
   EmployeeID int NOT NULL,
   ContactFirstNM varchar(50) NOT NULL,
   ContactLastNM varchar(50) NOT NULL,
   ContactPhoneNBR varchar(25) NOT NULL,
   CONSTRAINT UNQ_EmergencyContact_FirstNM_LastNM_PhoneNBR
   UNIQUE (ContactFirstNM, ContactLastNM, ContactPhoneNBR))
```

169

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在第一个示例中，在CREATE TABLE中为某个列定义了一个UNIQUE约束：

```
AnnualReviewSummaryDESC varchar(900) NOT NULL UNIQUE
```

UNIQUE关键字后面跟列定义，表示要在AnnualReviewSummaryDESC列上创建一个UNIQUE约束。

在第二个示例中，根据CREATE TABLE中定义的3个列创建了一个UNIQUE约束。约束在列定义之后定义。代码的第一行定义了约束名：

```
CONSTRAINT UNQ_EmergencyContact_FirstNM_LastNM_PhoneNBR
```

代码的第二行定义了约束类型（UNIQUE），并在圆括号内填写了组成约束的列列表：

```
UNIQUE (ContactFirstNM, ContactLastNM, ContactPhoneNBR)
```


4.5.2 为既有表增加 UNIQUE 约束

可以使用ALTER TABLE来为既有表增加UNIQUE约束。语法如下：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
UNIQUE (column [ ASC | DESC ] [ ,...n ] )
```

表4-12详细描述了此命令的参数。

表4-12 ALTER TABLE...ADD CONSTRAINT（唯一）参数

参 数	描 述
table_name	这个参数指定了接收新唯一键索引的表名
constraint_name	这个参数表示要增加的约束的唯一名
column [ASC \| DESC] [,...n]	存储在列中的值必须唯一标识表中的一行（比如，没有两行对于所有指定列可以有相同的值）。ASC（升序）和DESC（降序）选项定义了列在聚集和非聚集索引中的排列顺序

这个示例演示了为Production.Culture表增加一个UNIQUE键：

```
ALTER TABLE Production.Culture
ADD CONSTRAINT UNQ_Culture_Name
UNIQUE (Name)
```

170

解析

在这个示例中，代码的第一行定义了要修改的表：

```
ALTER TABLE Production.Culture
```

代码的第二行定义了约束的名字：

```
ADD CONSTRAINT UNQ_Culture_Name
```

代码的第三行定义了约束类型，后面跟的是要应用的列名：

```
UNIQUE (Name)
```

在UNIQUE约束定义中指定的列在表中不能有重复的值，否则操作会失败，并得到一个存在重复键值的错误。

4.5.3 使用 CHECK 约束

CHECK约束用于定义列允许的格式和值。CHECK约束的语法如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CHECK ( logical_expression )
```

如果CHECK的逻辑表达式计算为TRUE，行就会被插入。如果CHECK约束的表达式计算为FALSE，行插入就会失败。本例演示了向CREATE TABLE定义中增加一个CHECK约束。GPA列的值会被限制在某个数值范围之内：

```
-- 删除表的旧版本
DROP TABLE Person.EmployeeEducationType

CREATE TABLE Person.EmployeeEducationType(
   EmployeeEducationTypeID int NOT NULL PRIMARY KEY,
   EmployeeID int NOT NULL,
   EducationTypeID int NULL,
   GPA numeric(4,3) NOT NULL CHECK (GPA > 2.5 AND GPA <=4.0))
```

在前面的示例中，CHECK约束表达式是在列约束级别定义。CHECK约束还能定义在表约束级别——如下面的示例所示，允许在表达式中引用多个列：

```
-- 删除表的旧版本
DROP TABLE Person.EmployeeEducationType

CREATE TABLE Person.EmployeeEducationType(
   EmployeeEducationTypeID int NOT NULL PRIMARY KEY,
   EmployeeID int NOT NULL,
   EducationTypeID int NULL,
   GPA numeric(4,3) NOT NULL,
   CONSTRAINT CK_EmployeeEducationType
   CHECK (EducationTypeID > 1 AND GPA > 2.5 AND GPA <=4.0))
```

解析

在第一个示例中，对Person.EmployeeEducationType表的GPA列定义了CHECK列约束：

171

```
GPA numeric(4,3) NOT NULL CHECK (GPA > 2.5 AND GPA <=4.0)
```

表中只允许GPA列的值大于2.5且小于等于4.0——这个范围之外的任何值会导致INSERT或UPDATE失败。

在第二个示例中，CHECK表约束计算了两个表列：

```
CHECK (EducationTypeID > 1 AND GPA > 2.5 AND GPA <=4.0)
```

除了GPA的要求，这个CHECK约束还需要EducationTypeID值大于1。

4.5.4　为既有表增加 CHECK 约束

和其他约束类型相似，你可以使用ALTER TABLE和ADD CONSTRAINT为既有表增加一个CHECK约束。语法如下：

```
ALTER TABLE table_name
WITH CHECK | WITH NOCHECK
ADD CONSTRAINT constraint_name
CHECK ( logical_expression )
```

表4-13详细描述了此命令的参数。

表4-13　ALTER TABLE...ADD CONSTRAINT（检查）参数

参　数	描　述
table_name	这个参数指定了接收新CHECK约束的表名

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

参　数	描　述
CHECK \| WITH NOCHECK	使用CHECK选项（默认），既有数据会被新的CHECK约束验证。NOCHECK会跳过新数据的验证，限制约束验证新的值（插入的或者更新的）
constraint_name	这个参数定义了CHECK约束的名字
logical_expression	这个参数指定了用来限制列中允许的值的逻辑表达式

在这个示例中，一个新的CHECK请求增加到了Person.ContactType表中：

```
ALTER TABLE Person.ContactType WITH NOCHECK
ADD CONSTRAINT CK_ContactType
CHECK (Name NOT LIKE '%assistant%')
```

解析

新的约束被加到了Person.ContactType表中，不允许任何像assistant这样的名字。ALTER TABLE语句的第一部分包含了WITH NOCHECK：

```
ALTER TABLE Person.ContactType WITH NOCHECK
```

如果语句执行的时候没有带WITH NOCHECK，执行就会失败，因为在表中已经存在名字中具有assistant的行。增加WITH NOCHECK意味着既有值会被忽略，只有新的值才会使用CHECK约束进行验证。

警告 使用WITH NOCHECK可能会导致以后发生问题，因此你不能依赖表中遵从约束的数据。 172

语句的后面一部分定义了新的约束名：

```
ADD CONSTRAINT CK_ContactType
```

在约束类型CHECK之后跟的是限制Name列内容的逻辑表达式：

```
CHECK (Name NOT LIKE '%assistant%')
```

4.5.5 禁止和启用约束

前面的示例演示了向表加入新约束的时候使用NOCHECK忽略违反新的约束规则的既有值。约束用于维持数据的完整性，但是有的时候你可能会需要取消这个规则来进行一些一次性的数据导入工作或非标准的业务操作。NOCHECK还能用来禁止CHECK或FOREIGN KEY约束，允许你插入违反约束规则的行。

在开始这个示例之前，向Sales.PersonCreditCard表发出一个插入请求：

```
INSERT Sales.PersonCreditCard
(BusinessEntityID, CreditCardID)
VALUES (14425, 924533)
```

插入失败了，返回下面的错误消息：

```
消息547，级别16，状态0，第1行
INSERT语句与FOREIGN KEY约束'FK_PersonCreditCard_CreditCard_CreditCardID'冲突。
该冲突发生于数据库'AdventureWorks'，表'Sales.CreditCard',列'CreditCardID'。
语句已终止。
```

接下来，我们使用NOCHECK来禁用导致前面错误消息的外键约束：

```
ALTER TABLE Sales.PersonCreditCard
NOCHECK CONSTRAINT FK_PersonCreditCard_CreditCard_CreditCardID
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

然后再次尝试插入：

```
INSERT Sales.PersonCreditCard
(BusinessEntityID, CreditCardID)
VALUES (14425, 924533)
```

这次就成功了：

```
(1行受影响)
```

然后我们可以DELETE新插入的行，这样一旦约束再次启用，就不存在数据完整性问题：

```
DELETE Sales.PersonCreditCard
WHERE BusinessEntityID = 14425 AND
      CreditCardID = 924533
```

要重新启用外键约束的检查，则在ALTER TABLE语句中使用CHECK：

```
ALTER TABLE Sales.PersonCreditCard
CHECK CONSTRAINT FK_PersonCreditCard_CreditCard_CreditCardID
```

173

要为表禁用或启用所有CHECK和FOREIGN KEY约束，应该使用ALL关键字，如本例所演示的那样：

```
-- 禁用所有约束的检查
ALTER TABLE Sales.PersonCreditCard
NOCHECK CONSTRAINT ALL

-- 启用所有约束的检查
ALTER TABLE Sales.PersonCreditCard
CHECK CONSTRAINT ALL
```

警告 为表禁用所有CHECK和FOREIGN KEY约束的行为应该只在绝对必要的时候进行。并且在操作完成后重新启用所有约束。

解析

在本技巧中，我们尝试了向Sales.PersonCreditCard表插入一个不存在于主键表的CreditCardID。插入操作和FK_PersonCreditCard_CreditCard_CreditCardID外键约束产生了冲突。

为了禁止约束验证新的值，我们使用了ALTER TABLE和NOCHECK CONSTRAINT。在使用NOCHECK禁止了约束之后，即使CreditCardID不存在于主键表，也允许它插入。

下面的示例演示了使用ALL关键字禁止表所有的外键和检查约束：

```
NOCHECK CONSTRAINT ALL
```

然后使用下面的代码来为表重新启用所有约束：

```
CHECK CONSTRAINT ALL
```

4.5.6 在表创建时使用 DEFAULT 约束

如果不知道行首次插入到表的时候列的值，就可以使用DEFAULT约束，用一个预期的或者非空的值来填充那个列。在CREATE TABLE的列定义中指定默认值的语法如下：

```
DEFAULT constant_expression
```

constant_expression就是当列值没有在INSERT中显式指定的时候，希望填充到列的默认值。

本例演示了设置EducationTypeID列的默认值为1：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
--删除旧表
DROP TABLE Person.EmployeeEducationType

CREATE TABLE Person.EmployeeEducationType(
   EmployeeEducationTypeID int NOT NULL PRIMARY KEY,
   EmployeeID int NOT NULL,
   EducationTypeID int NOT NULL DEFAULT 1,
   GPA numeric(4,3) NOT NULL )
```

174

解析

在本例中，EducationTypeID的默认值被设置成1。DEFAULT关键字放在列定义之后，后面跟随默认值（必须符合列数据类型）：

```
EducationTypeID int NOT NULL DEFAULT 1
```

由于列有DEFAULT值，如果在INSERT语句中没有显式插入值，值1会被插入，而不是NULL值。

4.5.7 为既有表增加 DEFAULT 约束

和其他约束类型一样，你可以使用ALTER TABLE以及ADD CONSTRAINT为既有表的列增加默认约束。实现的语法如下：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
DEFAULT default_value
FOR column_name
```

表4-14详细描述了此命令的参数。

表4-14 ALTER TABLE...ADD CONSTRAINT（默认）参数

参 数	描 述
table_name	接收新的DEFAULT约束的表名
constraint_name	DEFAULT约束的名字
default_value	列的默认值
column_name	要使用默认值的列名

这个示例演示了为既有表的列增加默认值：

```
ALTER TABLE HumanResources.Company
ADD CONSTRAINT DF_Company_ParentCompanyID
DEFAULT 1 FOR ParentCompanyID
```

解析

在本例中，一个新的默认值被应用到既有表的列上。ALTER TABLE的第一行定义了被影响的表：

```
ALTER TABLE HumanResources.Company
```

语句的第二行增加了一个约束，并且定义了约束的名字：

```
ADD CONSTRAINT DF_Company_ParentCompanyID
```

代码的第三行定义了约束类型DEFAULT，后面跟的是使用的默认值：

```
DEFAULT 1
```

最后，FOR子句中使用了要应用默认值的列：

```
FOR ParentCompanyID
```

175

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

4.5.8　从表删除约束

刚才我们已经学习了几个能加到表上的约束，在这个技巧中，会演示如何使用ALTER TABLE和DROP CONSTRAINT删除一个约束。删除一个约束的基本语法如下：

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name
```

table_name指定了要删除约束的表，constraint_name指定了要删除的约束名。在这个示例中，我们从HumanResources.Company表删除一个默认约束：

```
ALTER TABLE HumanResources.Company
DROP CONSTRAINT DF_Company_ParentCompanyID
```

解析

本例中代码的第一行指定了要删除约束的表：

```
ALTER TABLE HumanResources.Company
```

代码的第二行指定了要删除的约束名：

```
DROP CONSTRAINT DF_Company_ParentCompanyID
```

注意，并不需要约束类型，只使用约束名就可以了。要找到表的约束名，可以使用sp_help系统存储过程。

4.6　临时表和表变量

临时表的定义与普通表类似，只是它们自动保存在tempdb数据库中（无论你在哪个数据库中创建它们）。临时表通常用于以下一些情况。

- 替代游标。例如，可以生成一个临时表来替代使用T-SQL游标的方式来遍历结果集。使用WHILE循环，能遍历表中的每一行，为某行执行操作，然后从临时表中删除该行。
- 作为结果集额外的存储空间。例如，假设有一个SELECT查询对10个表执行联结操作。有的时候有很多联结的查询会执行得很糟糕。可以尝试的一种技术是把大查询打散成小的、额外的查询。使用临时表，可以基于更小的查询来创建中间结果集，而不是执行一个非常大的、多重联结的查询。
- 作为一个临时的、低开销的查找表。例如，假设正在使用一个需要执行数秒，但只返回一个很小的结果集的查询。你会在存储过程的几个地方使用这个结果集，但每次你引用它都会带来查询执行的时间开销。要解决这个问题，可以在存储过程中只执行一次查询，填充临时表。然后你可以在代码的多个地方引用临时表，不会带来额外的开销。

有两种不同的临时表类型：全局（global）和本地（local）。本地临时表有一个#符号前缀，全局临时表有一个##符号前缀。

176

本地临时表对于创建它们的当前用户连接可用。多个连接能创建多个名字相同的临时表，而不会引起冲突。本地表在内部表现为一个唯一的名字，所以其他连接在tempdb数据库中创建的同名临时表不会有冲突。本地临时表可以通过DROP语句来删除，或者当用户连接关闭后自动从内存中删除。

全局临时表和本地临时表的作用域不同。一旦一个连接创建了全局临时表，任何对当前数据库有适当权限的用户都能访问表。和本地临时表不同，你不能创建相同版本的全局临时表，因为这样会产生命名冲突。如果全局临时表通过DROP TABLE被显示删除，则它会从SQL Server中移除。当创建它的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

连接退出并且临时表不再被其他连接引用后，它也会被自动移除。补充一句，我很少看到在实际中使用全局临时表。如果表必须跨连接共享，应该创建真实的表而不是全局临时表。尽管这样，微软还是多提供了一种选择。

因为性能问题，DBA们非常排斥临时表——某些抱怨是确有其事的，某些则不。临时表确实会给tempdb带来不必要的磁盘开销，在创建的时候锁住tempdb，并且如果包含在存储过程定义中的话还会引起存储过程重新编译（重新编译就是重新创建存储过程的执行计划而不是重用）。

表变量（table variable）是可以用于T-SQL批处理、存储过程或函数中的数据类型，其创建和定义的方式和表相似，只是需要严格定义生存周期。当数据集比较小时使用表变量比使用临时表更好。表变量的统计信息不会像普通表或临时表那样得到维护，因此使用太大的表变量可能会引发查询优化的问题。和普通表或临时表不同，表变量不能加索引或者FOREIGN KEY约束。但表变量允许在表定义中有一些约束（PRIMARY KEY、UNIQUE、CHECK）。

使用表变量的理由有如下几条。

- 有作用域。表变量的生存周期只是批处理、函数或存储过程的持续时间。
- 更短的锁时间段（因为严格的周期）。
- 在存储过程中使用，出现重编译的情况较少。

正如前面提到的，使用表变量也有缺点。如果结果集变得很大或列数据的基数已经威胁到查询优化过程时，会影响表变量的性能。当遇到性能问题的时候，确保测试了所有的可用解决方案，不要盲目确信一种选项（临时表）比另外一种（表变量）更差。

4.6.1 使用临时表在批处理中进行多次查找

在这个示例中，会演示创建本地临时表，它会在批处理查询中引用多次。如果查询需要花费几秒时间来查找值，那么这项技术非常有用。我们可以查询预存的临时表而不是执行SELECT查询多次：

```
CREATE TABLE #ProductCostStatistics
( ProductID int NOT NULL PRIMARY KEY,
  AvgStandardCost money NOT NULL,
  ProductCount int NOT NULL)

INSERT #ProductCostStatistics
(ProductID, AvgStandardCost, ProductCount)
SELECT ProductID,
      AVG(StandardCost) AvgStandardCost,
      COUNT(ProductID) Rowcnt
FROM Production.ProductCostHistory
GROUP BY ProductID

SELECT TOP 3 *
FROM #ProductCostStatistics
ORDER BY AvgStandardCost ASC

SELECT TOP 3 *
FROM #ProductCostStatistics
ORDER BY AvgStandardCost DESC

SELECT AVG(AvgStandardCost) Average_of_AvgStandardCost
FROM #ProductCostStatistics

DROP TABLE #ProductCostStatistics
```

这个查询从临时表中返回3个结果集：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

ProductID	AvgStandardCost	ProductCount
873	0.8565	1
922	1.4923	1
870	1.8663	1

ProductID	AvgStandardCost	ProductCount
749	2171.2942	1
750	2171.2942	1
751	2171.2942	1

Average_of_AvgStandardCost
423.0001

解析

在这个技巧中，我们创建了一个叫做#ProductCostStatistics的临时表。我们向它插入行，与对普通表的操作一样，然后查询临时表3次（也和普通表一样），然后删除临时表。使用和普通表相同的语法来创建和查询表，只是临时表的名字有#符号作为前缀。在原始查询需要很长时间来执行的情况下，这是一项可以考虑的技术。

4.6.2 创建表变量来保存临时结果集

178 在2.4节中就介绍过表变量。我们学了如何使用表变量来保存OUTPUT命令的结果。

注解 SQL Server 2008引入了表值参数和用户定义类型，可以用它来在模块之间传递临时结果集。第11章会讲到这些主题。

创建表变量的语法和创建表的相似，不同的是使用DECLARE关键字，并且表名有@符号前缀：

```
DECLARE @TableName TABLE
   (column_name <data_type> [ NULL | NOT NULL ] [ ,...n ] )
```

在这个示例中，使用表变量的方式和前面技巧中使用临时表的方式相似。这个示例演示实现的不同（包括不显式地DROP表的方法）：

```
DECLARE @ProductCostStatistics TABLE
( ProductID int NOT NULL PRIMARY KEY,
  AvgStandardCost money NOT NULL,
  ProductCount int NOT NULL)

INSERT @ProductCostStatistics
(ProductID, AvgStandardCost, ProductCount)
SELECT ProductID,
      AVG(StandardCost) AvgStandardCost,
      COUNT(ProductID) Rowcnt
FROM Production.ProductCostHistory
GROUP BY ProductID

SELECT TOP 3 *
FROM @ProductCostStatistics
ORDER BY ProductCount
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ProductID    AvgStandardCost    ProductCount
710          3.3963             1
709          3.3963             1
731          352.1394           1
```

解析

这个技巧使用表变量的方式和前面技巧使用临时表的方式基本一样。不过，两个技巧还是有重要区别的。

首先，这次的表变量使用DECLARFE @ Tablename TABLE而不是CREATE TABLE来定义。其次，和临时表技巧不同，该技巧在每个语句后面没有GO，因为临时表只能在批处理、存储过程或函数中存在。

在技巧的后面一部分中，像使用普通表一样从表变量插入和查询，不同的是这次使用@tablename格式：

```
INSERT @ProductCostStatistics
...

SELECT TOP 3 *
FROM @ProductCostStatistics
...
```

在示例的最后并不需要使用DROP TABLE，因为在批处理/存储过程/函数执行后表变量就从内存中消失了。

4.7 管理超大型表

接下去的一些技巧会演示管理超大型表（例如数百万的行）的一些方法。具体来说，我们将讨论SQL Server表分区功能和文件组的放置。

表分区提供了内建的方法来水平划分表和/或索引中的数据，同时要管理一个逻辑对象。水平分区是指每一个分区都有同样数量的列，只是减少了行的数量。分区能使超大型表和/或索引的管理变得简单，减少加载时间，改善查询时间，并允许更小的维护窗口。本部分接下来的几个技巧会演示如何使用T-SQL命令来创建、修改和管理分区以及划分数据库对象。

提示 SQL Server 2008增强了分区表查询处理功能，包括可识别分区查找操作、在执行计划中访问的分区更好的表现方式，以及新跟踪标志2440，它启用了在并行查询计划中为每个分区分配多线程执行的能力。

我也会讨论文件组（filegroup）放置。数据库数据文件属于文件组，每一个数据库有一个主要的文件组，需要的话我们可以加入另外的文件组。为数据库加入新的文件组通常用于超大型数据库（VLDB），因为它们通过把数据划分成几个阵列，能使备份管理操作变得简单，并潜在提升性能。我会在本章的最后一个技巧中演示把一个表放置在某个文件组中。

在接触和分区相关的技巧之前，我们先讨论两个新的命令CREATE PARTITION FUNCTION和CREATE PARTITION SCHEME。CREATE PARTITION FUNCTION根据某列的值来将列映射到分区。例如，如果你对某个datetime数据类型的列进行运算，可以根据年或月把数据划分到不同的文件组。

CREATE PARTITION FUNCTION的基本语法如下：

```
CREATE PARTITION FUNCTION partition_function_name(input_parameter_type)
AS RANGE [ LEFT | RIGHT ]
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FOR VALUES ( [ boundary_value [ ,...n ] ] )
```

表4-15详细描述了此命令的参数。

表4-15　CREATE PARTITION FUNCTION参数

参　数	描　述
partition_function_name	这个参数指定了分区函数名
input_parameter_type	这个参数指定了分区列的数据类型。你不能使用大值数据类型(text、ntext、image、xml、timestamp、varchar(max)、varbinary(max)、nvarchar(max))、CLR用户定义数据类型或者别名数据类型。如果你希望根据一个datetime列来划分表数据，可以为input_parameter_type指定datetime
LEFT \| RIGHT	你还有一个LEFT或RIGHT的选择。它定义了boundary_value参数定义的值属于边界的哪一侧（接下来的解析部分会讨论LEFT和RIGHT）
[boundary_value [,...n]]	这个参数定义每个分区的值域。你可以定义多达999个分区（然而由于性能问题，可能不推荐这样做）。你在这个参数中选择的n个值会产生$n+1$个分区（同样，在解析部分中会看到更深层次的解释）

180

一旦创建了分区函数，它就可以被一个或多个分区方案使用。分区方案把定义在分区函数中的分区映射到实际的文件组。

CREATE PARTITION SCHEME的基本语法如下：

```
CREATE PARTITION SCHEME partition_scheme_name
AS PARTITION partition_function_name
[ ALL ] TO ( { file_group_name | [PRIMARY] } [ ,...n] )
```

表4-16详细描述了此命令的参数。

表4-16　CREATE PARTITION SCHEME参数

参　数	描　述
partition_scheme_name	这个参数指定了分区方案的名字
partition_function_name	这个参数表示分区方案绑定的分区函数名
ALL	如果指定了ALL，所有分区会映射到file_group_name参数指定的文件组
{ file_group_name \| [PRIMARY] } [,...n]	这个参数定义了分配到每个分区的文件组。如果指定了PRIMARY，分区会存储于主文件组中

4.7.1　实现表分区

在这个技巧中，我会介绍以下方法。

- 创建一个或多个文件组来持有分区。
- 把文件加入在分区过程中使用的每个文件组。
- 使用CREATE PARTITION FUNCTION命令来决定表中的数据如何分区。
- 使用CREATE PARTITION SCHEME命令来绑定PARTITION FUNCTION到指定文件组。
- 创建表，绑定某个分区列到PARTITION SCHEME。

这个技巧创建一个叫Sales.WebSiteHits的表，它用来跟踪一个虚拟网站上的每个点击。在这种情况下，表很快会变得很大，查询可能会因此执行得很慢，而且对整个数据库的备份操作花费的时间比当前的维护窗体允许的时间更长。

为了解决这个应用程序的情况，表中的数据会被水平划分，也就是根据某个列(在这里是HitDate)把几组行映射到磁盘上不同的底层物理文件中。这个示例的第一部分演示为AdventureWorks数据库增

181

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

加新的文件组：

```
ALTER DATABASE AdventureWorks
ADD FILEGROUP hitfg1

ALTER DATABASE AdventureWorks
ADD FILEGROUP hitfg2

ALTER DATABASE AdventureWorks
ADD FILEGROUP hitfg3

ALTER DATABASE AdventureWorks
ADD FILEGROUP hitfg4
```

接着，对于创建的每个新文件组，向其中加入一个新的数据库文件：

```
ALTER DATABASE AdventureWorks
ADD FILE
(   NAME = awhitfg1,
    FILENAME = 'c:\Apress\aw_hitfg1.ndf',
    SIZE = 1MB
)
TO FILEGROUP hitfg1
GO

ALTER DATABASE AdventureWorks
ADD FILE
(   NAME = awhitfg2,
    FILENAME = 'c:\Apress\aw_hitfg2.ndf',
    SIZE = 1MB
)
TO FILEGROUP hitfg2
GO

ALTER DATABASE AdventureWorks
ADD FILE
(   NAME = awhitfg3,
    FILENAME = 'c:\Apress\aw_hitfg3.ndf',
    SIZE = 1MB
)
TO FILEGROUP hitfg3
GO

ALTER DATABASE AdventureWorks
ADD FILE
(   NAME = awhitfg4,
    FILENAME = 'c:\Apress\aw_hitfg4.ndf',
    SIZE = 1MB
)
TO FILEGROUP hitfg4
GO
```

现在，文件组已经为分区数据准备就绪了，我们将创建分区函数，决定表中的数据如何水平划分（在这里是根据日期范围）：

182

```
CREATE PARTITION FUNCTION HitDateRange (datetime)
AS RANGE LEFT FOR VALUES ('1/1/2006', '1/1/2007', '1/1/2008')
GO
```

在创建分区函数之后，创建分区方案来绑定分区函数到新的文件组：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE PARTITION SCHEME HitDateRangeScheme
AS PARTITION HitDateRange
TO ( hitfg1, hitfg2, hitfg3, hitfg4 )
```

最后，CREATE TABLE在ON子句使用HitDate列来创建一个使用这个分区方案的表：

```
CREATE TABLE Sales.WebSiteHits
(WebSiteHitID bigint NOT NULL IDENTITY(1,1),
 WebSitePage varchar(255) NOT NULL,
 HitDate datetime NOT NULL,
 CONSTRAINT PK_WebSiteHits
 PRIMARY KEY (WebSiteHitID, HitDate))
ON [HitDateRangeScheme] (HitDate)
```

解析

在技巧的第一部分中，向AdventureWorks数据库加入了4个新的文件组。之后，向每一个文件组加入了一个数据库文件。

接着，创建一个分区函数，它定义了分区函数的分区边界和期望的分区列数据类型。在CREATE PARTITION FUNCTION命令的第一行，我们选择了datetime数据类型：

```
CREATE PARTITION FUNCTION HitDateRange (datetime)
```

下面一行定义了分区函数的值域，通过年份来创建分区：

```
AS RANGE LEFT FOR VALUES ('1/1/2006', '1/1/2007', '1/1/2008')
```

你可以定义多达999个分区（然而由于潜在的性能问题，不推荐这么多的分区）。你选择*n*个值就会得到*n*+1个分区。你还可以选择LEFT或者RIGHT，它定义了所定义的值属于哪个边界。在这个技巧中，我们选择了LEFT。表4-17列出了本例中的分区边界（或者说行被放置到哪个分区）。

表4-17　LEFT边界

位　置　#	最小的时间	最大的时间
1	最小允许的时间	1/1/2006 00:00:00
2	1/1/2006 00:00:01	1/1/2007 00:00:00
3	1/1/2007 00:00:01	1/1/2008 00:00:00
4	1/1/2008 00:00:01	最大允许的时间

183

如果选择了RIGHT，分区边界会如表4-18列出的那样。

表4-18　RIGHT边界

位　置　#	最小的时间	最大的时间
1	最小允许的时间	12/31/2005 12:59:59
2	1/1/2006 00:00:00	12/31/2006 12:59:59
3	1/1/2007 00:00:00	12/31/2007 12:59:59
4	1/1/2008 00:00:00	最大允许的时间

一旦创建了分区函数，它就能用于一个或多个分区方案。分区方案把分区函数中定义的分区映射到实际的文件组中。新的分区方案的第一行定义了分区方案名：

```
CREATE PARTITION SCHEME HitDateRangeScheme
```

代码的第二行定义了分区方案绑定的分区函数（在前面步骤中创建的函数）：

```
AS PARTITION HitDateRange
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

TO子句按照分区序列的次序定义了哪些文件组映射到分区函数中定义的4个分区：

```
TO ( hitfg1, hitfg2, hitfg3, hitfg4 )
```

在创建了分区方案后，就能把它绑定到表。在CREATE TABLE语句的ON子句中（表定义的最后一行），指定了分区方案，并在圆括号内指定要分区的列：

```
CREATE TABLE Sales.WebSiteHits
(WebSiteHitID bigint NOT NULL IDENTITY(1,1),
 WebSitePage varchar(255) NOT NULL,
 HitDate datetime NOT NULL,
 CONSTRAINT PK_WebSiteHits
 PRIMARY KEY (WebSiteHitID, HitDate))
ON [HitDateRangeScheme] (HitDate)
```

注意，主键由WebSiteHitID和HitDate组成。分区的键列（HitDate）必须是主键的一部分。

Sales.WebSiteHits表现在被分区了——但工作方式还是像一个普通表那样。你不需要在SELECT、INSERT、UPDATE或DELETE语句中作任何特殊的操作。在后台，当添加数据后，系统会根据分区函数和方案把行插入到合适的文件组。

4.7.2 确定数据在分区中的位置

因为分区操作在后台发生，所以不能直接查询各个分区。要确定数据属于哪个分区，可以使用$PARTITION函数。

$PARTITION的语法如下：

```
$PARTITION.partition_function_name(expression)
```

表4-19详细描述了此命令的参数。

表4-19 $PARTITION函数参数

参 数	描 述
partition_function_name	用于划分表的分区函数名
expression	作为分区键的列

这个示例演示了如何使用这个函数。首先，向已分区的Sales.WebSiteHits表插入4行：

```
INSERT Sales.WebSiteHits
(WebSitePage, HitDate)
VALUES ('Home Page', '10/22/2007')

INSERT Sales.WebSiteHits
(WebSitePage, HitDate)
VALUES ('Home Page', '10/2/2006')

INSERT Sales.WebSiteHits
(WebSitePage, HitDate)
VALUES ('Sales Page', '5/9/2008')

INSERT Sales.WebSiteHits
(WebSitePage, HitDate)
VALUES ('Sales Page', '3/4/2000')
```

然后使用SELECT和$PARTITION函数查询表：

```
SELECT HitDate,
   $PARTITION.HitDateRange (HitDate) Partition
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FROM Sales.WebSiteHits
```

这个查询返回：

```
HitDate                 Partition
2000-03-04 00:00:00.000 1
2006-10-02 00:00:00.000 2
2007-10-22 00:00:00.000 3
2008-05-09 00:00:00.000 4
```

解析

这个技巧首先向已分区的Sales.WebSiteHits表插入了4行。插入的每一行的HitDate的年份都不同（为了演示这个函数）。

接着，使用$PARTITION函数对表执行查询：

```
SELECT HitDate,
   $PARTITION.HitDateRange (HitDate) Partition
FROM Sales.WebSiteHits
```

partition_function_name是前一个技巧中创建的函数名。在圆括号中的表达式是HitDate，它是用来划分数据的列。

$PARTITION函数根据分区函数对每一个HitDate进行计算，确定它保存在哪个分区中。这样就可以看到数据怎么跨不同分区存储。如果一个分区和其他相比不平衡，可以创建或移除既有的分区——下面我们会演示这两个函数。

185

4.7.3 增加新的分区

随着时间的推移，你可能会希望为已分区的表添加额外的分区（例如，可以为每一个新年创建一个新的分区）。要增加一个新的分区，可以使用ALTER PARTITION SCHEME和ALTER PARTITION FUNCTION命令。

要想为一个既有的分区函数创建新的分区，首先必须准备一个文件组来保存新分区的数据（新的或者已有的文件组都行）。第一步是使用ALTER PARTITION SCHEME来指定下一个分区文件组。

ALTER PARTITION SCHEME的语法如下：

```
ALTER PARTITION SCHEME partition_scheme_name
NEXT USED [ filegroup_name ]
```

表4-20详细描述了此命令的参数。

表4-20 ALTER PARTITION SCHEME参数

参 数	描 述
partition_scheme_name	这个参数指定了需要修改的分区方案名
NEXT USED [filegroup_name]	NEXT USED关键字把下一个文件组列入使用队列，让任何新的分区使用

在增加了下一个文件组的引用之后，使用ALTER PARTITION FUNCTION来创建（分割）新的分区（移除/合并分区也一样）。ALTER PARTITION FUNCTION的语法如下：

```
ALTER PARTITION FUNCTION partition_function_name()
{
    SPLIT RANGE ( boundary_value )
  | MERGE RANGE ( boundary_value )
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表4-21详细描述了此命令的参数。

表4-21 ALTER PARTITION FUNCTION参数

参 数	描 述
partition_function_name	这个参数指定了要从中添加或移除分区的分区函数名
SPLIT RANGE (boundary_value) \| MERGE RANGE (boundary_value)	SPLIT RANGE通过定义一个新的边界值来创建一个新的分区。MERGE RANGE用于移除一个既有的分区

这个示例演示如何创建（分割）一个新的分区。第一步就是新建一个为新分区所用的文件组。在本例中，使用PRIMARY文件组：

```
ALTER PARTITION SCHEME HitDateRangeScheme
NEXT USED [PRIMARY]
```

接着，修改分区函数来创建新的分区，定义边界为2009年1月1日：

```
ALTER PARTITION FUNCTION HitDateRange ()
SPLIT RANGE ('1/1/2009')
```

在新分区创建后，插入一个新行来测试新分区：

```
INSERT Sales.WebSiteHits
(WebSitePage, HitDate)
VALUES ('Sales Page', '3/4/2009')
```

使用$PARTITION来查询表：

```
SELECT   HitDate,
$PARTITION.HitDateRange (HitDate) Partition
FROM Sales.WebSiteHits
```

这个查询显示新插入的行已经存储到了新的分区中（分区号5）：

```
HitDate                 Partition
2000-03-04 00:00:00.000 1
2006-10-02 00:00:00.000 2
2007-10-22 00:00:00.000 3
2008-05-09 00:00:00.000 4
2009-03-04 00:00:00.000 5
```

解析

在这个技巧的示例中，使用ALTER PARTITION SCHEME和NEXT USED关键字来修改HitDateRangeScheme。NEXT USED关键字把下一个文件组加入队列，供新的分区使用。默认的PRIMARY文件组被选为新分区的目标存储区：

```
ALTER PARTITION SCHEME HitDateRangeScheme
NEXT USED [PRIMARY]
```

然后使用ALTER PARTITION FUNCTION和SPLIT RANGE来加入一个新的分区边界：

```
ALTER PARTITION FUNCTION HitDateRange ()
SPLIT RANGE ('1/1/2006')
```

添加新的分区只需要使用一个新值，其实是使用原来的边界类型（LEFT或RIGHT）把一个既有的分区范围分成了2份。一次分离只可以使用一次SPLIT RANGE——不可以在一条语句中增加多个分区。

这个示例的分割增加了一个新的分区#5，如表4-22所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表4-22 新的分区布局

位 置 #	最小的时间	最大的时间
1	最小允许的时间	1/1/2006 00:00:00
2	1/1/2006 00:00:01	1/1/2007 00:00:00
3	1/1/2007 00:00:01	1/1/2008 00:00:00
4	1/1/2008 00:00:01	1/1/2009 00:00:00
5	1/1/2009 00:00:01	最大允许的时间

向表Sales.WebSiteHits插入了新的一行，使用了分区函数。然后执行查询来查看每行所属的分区，
187 并确认了新行插入到了第5个分区中。

4.7.4 移除分区

前面的技巧介绍了ALTER PARTITION FUNCTION的语法，包括用于移除既有分区的MERGE RANGE功能的描述。移除一个分区本质上是把两个分区合并成一个，行重新分配到目标合并的分区中。

这个示例演示从HitDateRange分区函数中移除1/1/2007分区：

```
ALTER PARTITION FUNCTION HitDateRange ()
MERGE RANGE ('1/1/2007')
```

然后使用$PARTITION函数查询已分区的表：

```
SELECT   HitDate,
   $PARTITION.HitDateRange (HitDate) Partition
FROM  Sales.WebSiteHits
```

这个查询返回如下的结果：

```
HitDate                  Partition
2000-03-04 00:00:00.000  1
2007-10-22 00:00:00.000  2
2006-10-02 00:00:00.000  2
2008-05-09 00:00:00.000  3
2009-03-04 00:00:00.000  4
```

解析

ALTER PARTITION FUNCTION用来分割和合并分区。在这里，是使用MERGE RANGE关键字来删除1/1/2007分区边界：

```
ALTER PARTITION FUNCTION HitDateRange ()
MERGE RANGE ('1/1/2007')
```

然后执行查询来查看哪些行属于哪个分区。表4-23列出了MERGE之后的边界。

表4-23 新的分区布局

位 置 #	最小的时间	最大的时间
1	最小允许的时间	1/1/2006 00:00:00
2	1/1/2006 00:00:01	1/1/2008 00:00:00
3	1/1/2008 00:00:01	1/1/2009 00:00:00
4	1/1/2009 00:00:01	最大允许的时间

分区#2现在包含了2年的数据而不是1年。每一次执行ALTER PARTITION FUNCTION只可以合并一个

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

分区，并且不能使用ALTER PARTITION FUNCTION把一个已分区的表转化为一个不分区的表——只能把分区的数量减少到一个分区。

188

4.7.5 把分区移动到不同的表

使用SQL Server的分区功能，可以用最少的工作或最小开销在不同的表之间转移分区。可以使用ALTER TABLE...SWITCH在表之间转移分区。转移能以3种方式进行：从一个已分区的表转移一个分区到另外一个已分区的表（都需要对同一列进行分区），转移整个未分区的表到一个已分区的表，或者从一个已分区的表移动一个分区到一个未分区的表。

提示 在SQL Server 2005中，不能在绑定了架构的视图中引用分区表，也不允许使用索引视图。SQL Server 2008支持架构绑定和分区索引视图。

4

在表之间切换分区的基本语法如下：

```
ALTER TABLE tablename
SWITCH [ PARTITION source_partition_number_expression ]
TO [ schema_name. ] target_table
[ PARTITION target_partition_number_expression ]
```

表4-24详细描述了此命令的参数。

表4-24 ALTER TABLE..SWITCH参数

参 数	描 述
tablename	要移动分区的源表
source_partition_number_expression	要重新分配的分区号
[schema_name.] target_table	接收分区的目标表
partition.target_partition_number_expression	目标分区号

这个示例演示在Sales.WebSiteHits表和一个叫做Sales.WebSiteHitsHistory的新表之间移动分区。第一步是创建一个新表来保存历史网站点击信息：

```
CREATE TABLE Sales.WebSiteHitsHistory
(WebSiteHitID bigint NOT NULL IDENTITY(1,1),
 WebSitePage varchar(255) NOT NULL,
 HitDate datetime NOT NULL,
 CONSTRAINT PK_WebSiteHitsHistory
 PRIMARY KEY (WebSiteHitID, HitDate))
ON [HitDateRangeScheme] (HitDate)
```

然后，使用ALTER TABLE把Sales.WebSiteHits表的3号分区转移到新表Sales.WebSiteHitsHistory的3号分区中：

```
ALTER TABLE Sales.WebSiteHits SWITCH PARTITION 3
TO Sales.WebSiteHitsHistory PARTITION 3
```

接着，使用$PARTITION执行查询来查看转移到新表中的数据：

```
SELECT   HitDate,
$PARTITION.HitDateRange (HitDate) Partition
FROM  Sales.WebSiteHitsHistory
```

189

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
HitDate                 Partition
2008-05-09 00:00:00.000 3
```

解析

技巧的第一部分创建了一个叫做Sales.WebSiteHitsHistory的新表，并使用了和Sales.WebSiteHits表一样的分区方案。

源表和要转移的分区号在ALTER TABLE命令的第一行被引用：

```
ALTER TABLE Sales.WebSiteHits SWITCH PARTITION 3
```

TO关键字指定了目标表和要接受转移数据的目标表和分区：

```
TO Sales.WebSiteHitsHistory PARTITION 3
```

在表之间转移分区比手动执行行操作（例如INSERT...SELECT）快很多，因为并不是真正移动物理数据，只是修改了有关分区目前保存在哪里的元数据。同样要记住，任何既有表的目标分区必须是空的才能作为目的分区。如果它是一个未分区的表，它也必须是空的。

4.7.6 移除分区函数和分区方案

如果尝试删除一个绑定到既有表或索引的分区函数或分区方案，就会得到一个错误消息。也不能直接移除一个绑定到表的分区方案或分区函数（除非删除整个表，本技巧将会这么做）。如果开始是以堆方式创建表（没有聚集索引的表），然后创建一个聚集索引绑定到一个分区方案，那么可以使用CREATE INDEX DROP_EXISTING选项（参阅第5章）来重新创建索引而不引用分区方案。

使用下面的语法删除一个分区方案：

```
DROP PARTITION SCHEME partition_scheme_name
```

这个命令接受要删除的分区方案名作为参数。

使用下面的语法删除一个分区函数：

```
DROP PARTITION FUNCTION partition_function_name
```

同样，这个命令只接受要删除的分区函数名作为参数。

这个示例演示了如何删除一个分区函数和分区方案，假设在这种情况下可以删除源表（在正式应用下通常不能这样做）：

```
DROP TABLE Sales.WebSiteHitsHistory
DROP TABLE Sales.WebSiteHits

-- 删除分区方案和分区函数
DROP PARTITION SCHEME HitDateRangeScheme
DROP PARTITION FUNCTION HitDateRange
```

190

解析

这个示例演示了删除一个分区方案和分区函数。对于本例，我们需要在这之前先删除源表。

另外一种解决方案就是把所有的结果复制到一个外部表，删除表，删除分区方案和分区函数，然后重新命名把数据复制过去的那张表。如果你的目标只是把表变成一个分区，其实可以合并所有分区，并且保留分区方案和分区函数。只有一个分区的表在功能上和一个普通的未分区表一样。

4.7.7 用文件组提高 VLDB 的可管理性

文件组通常用于超大型数据库（VLDB），因为通过把数据划分成几个阵列，它们能使备份管理操

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

作变得简单，并潜在提升性能。当创建表的时候，可以指定它创建在某个文件组上。例如，如果你知道某个表会变得很大，可以指定它创建在某个文件组上。

注解 这个技巧包含了一些文件组技术和概念，第22章会有更详细的介绍。

指定一个表的文件组的基本语法如下：

```
CREATE TABLE ...
    [ ON {filegroup | "default" }]
[ { TEXTIMAGE_ON { filegroup | "default" } ]
```

表4-25详细描述了此命令的参数。

表4-25 在文件组上创建表的参数

参 数	描 述
filegroup	这个参数指定了创建表的文件组的名字
"DEFAULT"	设置表创建到数据库定义的默认文件组上
TEXTIMAGE_ON { filegroup \| "DEFAULT" }	这个选项允许text、ntext、image、xml、varchar(max)、nvarchar(max)、varbinary(max)数据类型的数据存储在独立的文件组中

这个示例演示如何把表放到非默认的、用户创建的文件组中。第一步需要在AdventureWorks数据库中创建一个新的文件组：

```
ALTER DATABASE AdventureWorks
ADD FILEGROUP AW_FG2
GO
```

然后向文件组加入新的文件：

```
ALTER DATABASE AdventureWorks
ADD FILE
(   NAME = AW_F2,
    FILENAME = 'C:\Apress\aw_f2.ndf',
    SIZE = 1MB
)
TO FILEGROUP AW_FG2
GO
```

然后把新表创建在新的文件组中（因此数据会保存在文件组中的新文件中）：

```
CREATE TABLE HumanResources.AWCompany(
   AWCompanyID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
   ParentAWCompanyID int NULL,
   AWCompanyNM varchar(25) NOT NULL,
   CreateDT datetime NOT NULL DEFAULT (getdate())
) ON AW_FG2
```

在第二个示例中，创建的表指定大值对象数据列保存在与普通数据不同的文件组（AW_FG2）中（普通数据保存在PRIMARY文件组中）：

```
CREATE TABLE HumanResources.EWCompany(
   EWCompanyID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
   ParentEWCompanyID int NULL,
   EWCompanyName varchar(25) NOT NULL,
   HeadQuartersImage varbinary(max) NULL,
   CreateDT datetime NOT NULL DEFAULT (getdate())
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
) ON [PRIMARY]
TEXTIMAGE_ON AW_FG2
```

解析

这个技巧从创建一个新的AW_FG2文件组开始。使用ALTER DATABASE命令可以完成这个操作。之后，向AdventureWorks数据库添加一个新的数据库文件，并把它放到新的文件组中。

然后像平时一样执行CREATE TABLE，只是在表定义的最后使用ON AW_FG2来把表放入新的AW_FG2文件组：

```
ON AW_FG2
```

如果没有在CREATE TABLE中使用ON文件组子句，那么表会放到默认文件组中（如果你没有修改它，它应该叫做PRIMARY）。

如果表变得很大，并且你已经把它放到了它的文件组中，就可以使用文件组备份来特别备份这个表或者任何其他放到其中的表或索引（更多有关把索引放到文件组中的信息请参阅第5章，要回顾文件组备份请参阅第29章）。

对于第二个示例，我们使用文件组选项创建表，把普通数据放到PRIMARY文件组中，把text/image数据放到AW_FG2文件组中（只有当你的表有大值数据类型时才可以这么做）：

```
ON [PRIMARY]
TEXTIMAGE_ON AW_FG2
```

把大值对象数据分离出来可以让数据库维护变得更简单，并且也可以提高性能，这取决于你的数据库设计和物理硬件、查询访问的类型和文件组中文件的位置。

4.7.8　用数据压缩减少磁盘空间使用

SQL Server 2008 企业版和开发版引入了表、索引和相应分区的行级和页级压缩。

192 行压缩为数值数据类型（例如，int、bigint和decimal）和固定长度类型（比如money和datetime）使用可变长度的存储。行压缩也为定长字符串使用可变长度存储格式，且不存储尾部的空字符、NULL和0值。

页压缩包含行压缩，以及前缀压缩和字典压缩。前缀压缩包括在行的列中存储多次的列前缀值的存储，并引用单个的值替换掉多余的前缀。字典压缩发生在前缀压缩后，首先查找数据页上任意位置的重复数据值（不只是前缀），然后使用指向单个值的指针替换多余的重复值。

提示　第5章将讲解如何使用CREATE INDEX和ALTER INDEX为非聚集索引启用压缩。

这个技巧将展示如何使用CREATE TABLE和ALTER TABLE来启用行压缩与页压缩。在第一个示例中，将为新表启用行压缩。指定DATA_COMPRESSION表选项并从NONE、ROW或PAGE中选择其一来完成它。

```
CREATE TABLE dbo.ArchiveJobPosting
(JobPostingID int NOT NULL IDENTITY(1,1) PRIMARY KEY CLUSTERED,
 CandidateID int NOT NULL,
 JobDESC char(2000) NOT NULL
)
WITH (DATA_COMPRESSION = ROW)
```

执行ALTER TABLE...REBUILD WITH来重新配置既有表的压缩选项。例如，如下命令关闭了我刚刚创建的表的压缩选项：

```
ALTER TABLE dbo.ArchiveJobPosting
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
REBUILD WITH
(DATA_COMPRESSION = NONE)
```

下一步，将向表中添加垃圾数据，演示压缩带来的好处。如下的查询插入一行，为CandidateID选择一个随机整型值，为JobDESC重复字母“a”50次。GO命令后跟100000的意思是INSERT将执行100 000次，从而100 000条新行插入到这个表中（如果你自己测试SQL Server实例，执行它可能会花费几分钟时间）：

```
INSERT dbo.ArchiveJobPosting
(CandidateID, JobDESC)
VALUES (CAST(RAND() * 10 as int),
        REPLICATE('a',50))
GO 100000
```

现在数据添加好了，可以执行sp_estimate_data_compression_saving系统存储过程来估计当使用行压缩或页压缩时可以节省多少磁盘空间。sp_estimate_data_compression_savings存储过程有5个参数：要压缩的表的架构名称、对象名称、索引ID、分区号和数据压缩方式（NONE、ROW或PAGE）。在如下的示例中，先检查使用行压缩可以节省多少空间：

```
EXEC sys.sp_estimate_data_compression_savings
      @schema_name = 'dbo',
      @object_name = 'ArchiveJobPosting',
      @index_id = NULL,
      @partition_number = NULL,
      @data_compression = 'ROW'
```

这个查询返回如下信息（为了便于阅读，调整了格式）：

object_name	ArchiveJobPosting
schema_name	dbo
index_id	1
partition_number	1
size_with_current_compression_setting(KB)	200752
size_with_requested_compression_setting(KB)	6536
sample_size_with_current_compression_setting(KB)	39776
sample_size_with_requested_compression_setting(KB)	1296

正如你在存储过程的结果中看到的，对当前的数据集增加的行压缩可以节省194 216KB。样本大小数据基于存储过程将样本数据载入到tempdb的克隆表中，并验证相应的压缩率。

现在将测试，查看使用页级压缩是否带来好处：

```
EXEC sys.sp_estimate_data_compression_savings
      @schema_name = 'dbo',
      @object_name = 'ArchiveJobPosting',
      @index_id = NULL,
      @partition_number = NULL,
     @data_compression = 'PAGE'
```

这个查询返回：

object_name	ArchiveJobPosting
schema_name	dbo
index_id	1
partition_number	1
size_with_current_compression_setting(KB)	200752
size_with_requested_compression_setting(KB)	1200
sample_size_with_current_compression_setting(KB)	40144
sample_size_with_requested_compression_setting(KB)	240

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

非常肯定，页级压缩比使用行级压缩会带来更大的好处。

警告 作为代价，压缩通常会增加CPU的使用。你必须考虑并测试你当前的应用程序，在磁盘空间和带来的CPU开销之间作出权衡。

然后，继续使用ALTER TABLE来启用表的页级压缩：

```
ALTER TABLE dbo.ArchiveJobPosting
REBUILD WITH
(DATA_COMPRESSION = PAGE)
```

也可以在分区级别配置压缩。在下面一组命令中，我将创建新的分区函数和方案，然后将其应用
194 到新表上。表会基于分区使用不同的压缩级别。首先创建分区函数和方案：

```
CREATE PARTITION FUNCTION pfn_ArchivePart(int)
AS RANGE LEFT FOR VALUES (50000, 100000, 150000)
GO

-- 这个命令假定数据已经有这些文件组
CREATE PARTITION SCHEME psc_ArchivePart
AS PARTITION pfn_ArchivePart
TO (hitfg1, hitfg2, hitfg3, hitfg4) ;
GO
```

下一步，创建在JobPostingID整型列上引用了分区方案的表。还要指定哪些分区将使用PAGE压缩，哪些分区使用行压缩：

```
CREATE TABLE dbo.ArchiveJobPosting_V2
(JobPostingID int NOT NULL IDENTITY(1,1) PRIMARY KEY CLUSTERED,
 CandidateID int NOT NULL,
 JobDESC char(2000) NOT NULL)
ON psc_ArchivePart(JobPostingID)
WITH (DATA_COMPRESSION = PAGE ON PARTITIONS (1 TO 3),
      DATA_COMPRESSION = ROW ON PARTITIONS (4))
```

如果希望为任一分区更改压缩级别，可以使用ALTER TABLE，如下面所演示的，将分区4从行压缩更改为页压缩：

```
ALTER TABLE dbo.ArchiveJobPosting_V2
REBUILD PARTITION = 4
WITH (DATA_COMPRESSION = PAGE)
```

解析

这个技巧演示了如何使用CREATE TABLE和ALTER TABLE为表应用页级和行级压缩。Sql Server 2008企业版和开发版引入了压缩特性，这个特性可以减少数据库表的总的磁盘使用空间。依赖于存储在表中数据的类型，整体压缩率将会有很大的不同。也要注意到，压缩在带来好处的同时，也会带来更大的CPU开销，因此需要在部署到生产运营环境之前进行彻底的测试。

将CREATE TABLE或ALTER TABLE命令与DATA_COMPRESSION子句结合使用就可以启用压缩（我将在第5章演示非聚集索引的压缩）。可以对堆（无聚集的索引）、聚集索引、非聚集索引、索引视图及表或索引指定的分区设置压缩。要验证增加行压缩或页压缩带来的好处，使用sp_estimate_data_compression_
195 saving系统存储过程，如本技巧中演示的。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第5章 索 引



索引可以帮助查询快速地访问存储在表和视图中的数据。使用索引，可以顺序地访问经过排序的数据行中的数据。这些行根据存储在某些列中的值进行排序。而其中列构成了索引键列，它们的值（对于任意给定的行）是一行的索引键。

本章涉及创建、修改以及删除不同类型的索引的技巧。我将演示如何创建索引，包括索引选项的语法、支持分区方案、INCLUDE命令、禁用页和行锁、禁用索引以及实施在线操作的能力。

我也将复习SQL Server 2008中的两个新特性，包括筛选索引与索引压缩。在进行本章的练习之前，建议你预先备份数据库AdventureWorks，这样你可以在练习这些技巧之后将它恢复到原始的状态。

注解 关于索引维护、重组织和重建（ALTER INDEX）的内容，参见第23章。索引视图的内容参见第7章。索引性能的故障解决与碎片整理参见第28章。

5.1 索引预览

索引是在表上创建的数据库对象，它可以提供到数据的更快的访问通道，并且可以使查询执行更快。索引为SQL Server提供了更有效地访问数据的方式。使用索引，你不用总去查找表中的每个数据页，检索特定行时也不用读取表的所有内容。

在默认情况下，常规的未作索引的表中的行不会以任何特定的顺序存储。处于无序状态的表被称为堆（heap）。为了基于一组搜索条件从堆中检索匹配的行，SQL Server不得不检查表中所有的行。即使只有一行匹配搜索条件，并且那行恰恰是SQL Server数据库引擎读取的第一行，SQL Server也仍然需要遍历每一条表记录，因为要知道是否有其他匹配行存在没有其他方法。这种信息扫描称为全表扫描。对于一个大型表，这可能意味着只是检索一行就要读取数百、数千、数百万或数十亿行。但是，如果SQL Server知道在表的一列或多列上存在索引，它就可能会使用这个索引更有效地去搜索匹配的记录。

在SQL Server中，表包含在一个或多个分区（partition）中。一个分区是一个组织单元，允许你水平划分表中和（或）索引中的数据，但仍然管理一个逻辑对象。当创建一个表时，默认情况下，它的所有数据都包含在一个分区中。分区包含了堆，或者当创建索引时，为B树结构（B-tree structure）。

当创建索引时，索引键数据存储在B树结构内。B树结构从索引开始的根节点开始。这个根节点（root node）拥有包含一系列指向下一级索引节点的索引键值的索引数据，称为中间叶级别（intermediate leaf level）。节点的底部级别称为叶级别（leaf level）。叶级别基于实际的索引类型是聚集（clustered）或非聚集（nonclustered）而不同。如果它是聚集索引，则叶级别是实际的数据页本身。如果为非聚集索引，则叶级别包含指向堆或聚集索引数据页的指针。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

聚集索引决定实际表数据如何进行物理存储。你只能指定一个聚集索引。这个索引类型依照指定的索引键列存储数据。图5-1演示了聚集索引的B树结构。注意叶级别是实际的数据页。

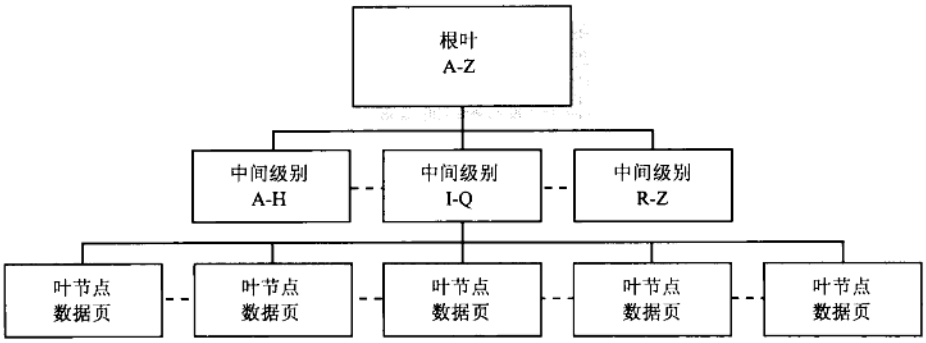


图5-1　聚集索引的B树结构

聚集索引的选择很关键，因为一个表只能拥有一个聚集索引。一般情况下，由于聚集索引的数据在物理上是以特定的顺序组织的，所以聚集索引经常选用以范围查询来查询的列。范围查询使用关键字BETWEEN以及大于（>）和小于（<）运算符。要考虑的其他列是那些用来排列大型结果集的列、用在聚合函数中的列以及包含完整唯一值的列。频繁更新的列以及非唯一列通常不是聚集索引键的好选择，因为聚集索引键包含在所有依赖的非聚集索引的叶级别上，这会引起额外的重新排序和修改。由于这个原因，你也应该避免在非常多或非常宽（许多字节的）的索引键上创建聚集索引。

非聚集索引存储在与物理数据分离的索引页上，指针指向位于索引页和节点的物理数据。非聚集索引列按照索引键列值的顺序进行存储。你可以在表上或索引视图上拥有最多249个非聚集索引。对于非聚集索引，叶节点级别是连接到指向堆的行或聚集索引行键的行定位器的索引键，如图5-2中所展示。

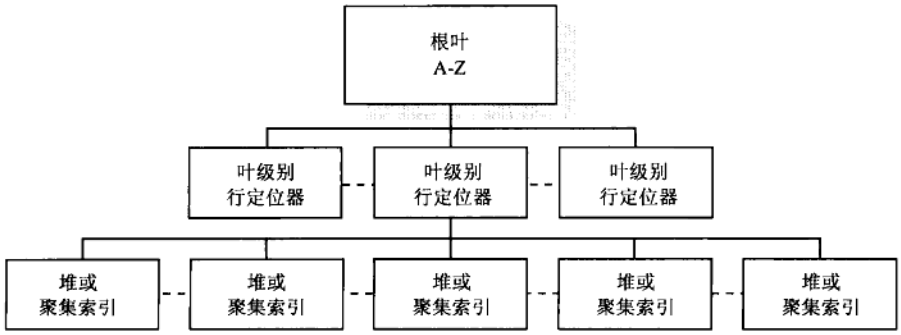


图5-2　非聚集索引的B树结构

当选择用在非聚集索引的列时，要查找在WHERE、JOIN以及ORDER BY子句中频繁引用的那些列。搜索返回较小结果集的高选择性列（少于表中所有行的20%）。选择性是指唯一索引键值对应多少行。如果列是低选择性的，例如只包含0或1，SQL Server不可能在创建查询执行计划时利用该查询，因为它的选择性很低。

198

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

不管是聚集索引还是非聚集索引，都基于一个或多个键值。索引键是指用来定义索引本身的列。使用本章稍后演示的新的INCLUDE子句，可以让非键列加到索引的叶级别上，这是SQL Server 2005的新特性。这个新特性允许你的查询选择的更多列被返回或被单个非聚集索引“覆盖”，从而减少总的I/O，因为SQL Server根本不需要访问聚集叶级别数据页。

一个索引中最多可以使用16个键列，只要所有索引键列组合起来不超过900字节就可以。不可以在索引键中使用大值对象数据类型，包括varchar(max)、nvarchar(max)、varbinary(max)、xml、ntext、text以及image数据类型。

聚集或非聚集索引可以被指定为唯一的或非唯一的。选择唯一索引需要确保插入到键列的数据值是唯一的。对于使用多个键的唯一索引（称为组合索引），键值组合在表中每行都必须是唯一的。

正如之前提到的，索引为提高查询性能提供很大的帮助，但也会带来相应的代价。你应该只根据预期的查询活动添加索引，并且必须持续地监视索引是否仍被使用。如果没有，应该将它们删除。在表上执行数据修改时，表上有太多的索引会引起性能开销，因为SQL Server除了数据的改变还必须维护索引的改变。正在进行的维护活动，比如索引重建和索引重组织，也会因为过多的索引造成延时。

随后的几个技巧将演示如何创建、修改、禁用、查看以及删除索引。

注解 要学习如何查看查询中使用了哪些索引，参见第28章。本章也会讲到如何查看索引碎片和标识出索引是否被使用。要学习如何重建和重组织索引，参见第23章。

5.1.1 创建表索引

这一技巧将向你展示如何创建两种类型的索引，一个是聚集的，另一个是非聚集的。通过使用CREATE INDEX命令创建索引。本章将回顾这个命令的多个方面，在下面讲到的示例中使用的基本语法如下所示：

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
    ON {
    [ database_name. [ schema_name ] . | schema_name. ]
        table_or_view_name}
( column [ ASC | DESC ] [ ,...n ] )
```

表5-1描述了这个命令的参数。

表5-1 CREATE INDEX命令的参数

参 数	描 述
[UNIQUE]	在每个表上只可以有一个主键。但是，如果你希望强制其他非键列是唯一的，可以为创建的索引指定UNIQUE约束。可以在单个表上创建多个UNIQUE索引，并且可以包括那些包含NULL值（尽管在每个列组合上只允许一个NULL值）的列
[CLUSTERED \| NONCLUSTERED]	这个参数指定索引类型，CLUSTERED或NONCLUSTERED。你只可以拥有一个CLUSTERED索引，但最多可以拥有249个NONCLUSTERED索引
index_name	这个参数定义了新索引的名称
[database_name. [schema_name] . \| schema_name.] table_or_view_name}	这个参数表示要索引的表或视图
column	这个参数指定用作索引键部分的一个或多个列
[ASC \| DESC]	这个参数指定了索引的特定列顺序，ASC为升序，DESC为降序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

我也将展示使用ALTER INDEX命令修改既有索引的一些示例：

```
ALTER INDEX index_name
ON object_name
...
```

这个命令包括多个与CREATE INDEX相同的选项，只是你不能使用它来改变使用哪些列以及它们的顺序。这个命令也用来重建或重组织索引（第23章会讲这些内容）

开始这个示例，在数据库AdventureWorks中创建新的表。我故意将PRIMARY KEY从表定义中去掉：

```
USE AdventureWorks
GO

CREATE TABLE HumanResources.TerminationReason(
   TerminationReasonID smallint IDENTITY(1,1) NOT NULL,
    TerminationReason varchar(50) NOT NULL,
   DepartmentID smallint NOT NULL,
 CONSTRAINT FK_TerminationReason_DepartmentID
   FOREIGN KEY (DepartmentID) REFERENCES
   HumanResources.Department(DepartmentID)
)
GO
```

200

在演示如何使用CREATE INDEX之前，记住当使用CREATE TABLE或ALTER TABLE在列上创建主键时，主键也会创建索引，这一点很重要。在这个示例中，我们先不这样做，而是使用带有ADD CONSTRAINT的ALTER TABLE在TerminationReasonID上创建一个CLUSTERED索引：

```
ALTER TABLE HumanResources.TerminationReason
ADD CONSTRAINT PK_TerminationReason PRIMARY KEY CLUSTERED (TerminationReasonID)
```

下一步，在DepartmentID列上创建非聚集索引：

```
CREATE NONCLUSTERED INDEX NCI_TerminationReason_DepartmentID ON
HumanResources.TerminationReason (DepartmentID)
```

解析

在这个练习中，创建了没有定义主键的表TerminationReason，意思是在最初的状态，这个表是一个“堆”。然后使用ALTER TABLE添加主键。在PRIMARY KEY语句后是CLUSTERED参数，这样就使用新的约束创建了聚集索引：

```
ALTER TABLE HumanResources.TerminationReason
ADD CONSTRAINT PK_TerminationReason PRIMARY KEY CLUSTERED (TerminationReasonID)
```

如果TerminationReasonID列没有选为主键，你仍然可以使用CREATE INDEX为其创建聚集索引：

```
CREATE CLUSTERED INDEX CI_TerminationReason_TerminationReasonID ON
HumanResources.TerminationReason (TerminationReasonID)
```

如果表上已经存在非聚集索引，创建新聚集索引会引发非聚集索引的重建，因为要使用聚集键交换非聚集叶级别行标识符。

示例中创建的非聚集索引如下：

```
CREATE NONCLUSTERED INDEX NCI_TerminationReason_DepartmentID ON
HumanResources.TerminationReason (DepartmentID)
```

两种索引的语法唯一不同的地方是位于CREATE和INDEX之间的参数NONCLUSTERED。

5.1.2 在非键列上强制唯一性

这一技巧将展示如何在非键表列上强制唯一性。前面技巧的CREATE INDEX的语法展示了UNIQUE关

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

键字。这个示例展示如何在表HumanResources.TerminationReason的TerminationReason列上创建唯一索引：

```
CREATE UNIQUE NONCLUSTERED INDEX UNI_TerminationReason ON
HumanResources.TerminationReason (TerminationReason)
```

现在，成功将两个新行插入到表中：

```
INSERT HumanResources.TerminationReason
(DepartmentID, TerminationReason)
VALUES (1, 'Bad Engineering Skills')

INSERT HumanResources.TerminationReason
(DepartmentID, TerminationReason)
VALUES (2, 'Breaks Expensive Tools')
```


如果尝试插入一行重复的、已经存在的TerminationReason值，将引发错误：

```
INSERT HumanResources.TerminationReason
(DepartmentID, TerminationReason)
VALUES (2, 'Bad Engineering Skills')
```

它返回：

```
消息2601，级别14，状态1，第9行
不能在具有唯一索引'UNI_TerminationReason'的对象
'HumanResources.TerminationReason'中插入重复键的行。
语句已终止。
```

从表中选择当前行，显示只有前两行被插入：

```
SELECT TerminationReasonID, TerminationReason, DepartmentID
FROM HumanResources.TerminationReason
```

它返回：

```
TerminationReasonID   TerminationReason          DepartmentID
1                     Bad Engineering Skills     1
2                     Breaks Expensive Tools     2
```

解析

在TerminationReason列上创建了唯一索引，这意味着每一行都必须有唯一值。你可以对一个表选择多个唯一约束。NULL值在唯一索引中是允许的，但是，它们只可以出现一次。像主键一样，唯一索引通过确认行可以被唯一标识来强制实体完整性。

5.1.3 在多个列上创建索引

这一技巧将展示如何创建多列索引。在前面的技巧中，我已经展示了如何在一个列上创建索引，但更多的时候你会希望更多列用在单个索引上。当两列或更多列经常在同一查询中被搜索或用于互相关联时，要使用组合索引。

在这个示例中，假定TerminationReason和DepartmentID经常用在同一SELECT查询的WHERE子句中。创建如下的NONCLUSTERED INDEX：

```
CREATE NONCLUSTERED INDEX NI_TerminationReason_TerminationReason_DepartmentID
ON HumanResources.TerminationReason(TerminationReason, DepartmentID)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

选择哪些列作为索引可以说是一种艺术。你会希望将索引添加到你认为经常会查询到的列；但是，必须牢记列的选择性。如果列的选择性很低，例如，在上千行中只包含几个唯一值，那么当创建查询
202 执行计划时，SQL Server不太可能利用此查询。创建组合索引时，一个经验做法是将最常选择的列放在开始部分，随后才是其他较少选择的列。在这个技巧的示例中，选择TerminationReason作为第一列，随后是DepartmentID。两个列都保证在表中是完全唯一的，所以它们具有相同的选择性。

提示 如果你刚接触到SQL Server，或为性能低下的查询寻找帮助，请使用数据库引擎优化顾问来帮助你生成基于查询或一批查询的索引建议。更多关于索引的使用和性能的内容请看第28章。

5.1.4 定义索引列排序方向

这个技巧将展示如何设置索引列的排序方向。索引列的默认排序方向是升序的。可以在CREATE INDEX的列定义中显式地使用ASC或DESC设置排序方式：

```
( column [ ASC | DESC ] [ ,...n ] )
```

在这个示例中，添加新列到表中，并且以降序排列索引：

```
ALTER TABLE HumanResources.TerminationReason
ADD ViolationSeverityLevel smallint
GO

CREATE NONCLUSTERED INDEX NI_TerminationReason_ViolationSeverityLevel
ON HumanResources.TerminationReason (ViolationSeverityLevel DESC)
```

解析

在这个技巧的示例中，新列ViolationSeverityLevel添加到了表TerminationReason中：

```
ALTER TABLE HumanResources.TerminationReason
ADD ViolationSeverityLevel smallint
GO
```

查询的编写者可能会希望对这个列值进行最常规的排序，从高到低显示ViolationSeverityLevel。你认为用户将在查询中如何使用ORDER BY，以此来匹配索引顺序，这样可以提升查询性能。在列名称后使用DESC指令创建索引：

```
(ViolationSeverityLevel DESC)
```

如果在你的索引中拥有多个键列，每一个都可以拥有它单独的排列顺序。

5.1.5 查看索引元数据

这个技巧将展示如何查看关于索引的有用的信息。在表上创建索引后，需要一些机制来跟踪它们位于何处、它们的名称是什么、类型以及定义它们的列。可以使用系统存储过程sp_helpindex来查看索引名称、描述以及特定表上的索引的键。这个系统存储过程只有一个参数，即要查看的索引所在的
203 表的名称。

这个示例演示了查看表Employee上的所有索引：

```
EXEC sp_helpindex 'HumanResources.Employee'
```

它返回如下部分结果：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
index_name                  index_description                                  index_keys
AK_Employee_LoginID         nonclustered, unique located on PRIMARY            LoginID
AK_Employee_NationalIDNumber nonclustered, unique located on PRIMARY           NationalIDNumber
AK_Employee_rowguid         nonclustered, unique located on PRIMARY            rowguid
IX_Employee_ManagerID       nonclustered located on PRIMARY                    ManagerID
PK_Employee_EmployeeID      clustered, unique, primary key located on PRIMARY  EmployeeID
```

如果要进行更深层次的索引分析，可以使用系统目录视图sys.indexes。例如如下的查询显示了表HumanResources.Employee的索引选项（本章稍后会讲到）：

```
SELECT SUBSTRING(name, 1,30) index_name,
      allow_row_locks,
      allow_page_locks,
      is_disabled,
      fill_factor,
      has_filter
FROM sys.indexes
WHERE object_id = OBJECT_ID('HumanResources.Employee')
```

它返回：

```
index_name                  allow_row_locks  allow_page_locks  is_disabled  fill_factor  has_filter
PK_Employee_EmployeeID      1                1                 0            0            0
AK_Employee_LoginID         1                1                 0            0            0
AK_Employee_NationalIDNumber 1               1                 0            0            0
AK_Employee_rowguid         1                1                 0            0            0
IX_Employee_ManagerID       1                1                 0            0            0
```

解析

可以调用系统存储过程sp_helpindex来列出特定表的索引。输出也返回索引的描述，包括类型和文件组位置。也会列出索引定义的键列。

系统目录视图sys.indexes也可以用来查出特定索引更多的配置设置情况。

提示 使用sys.index_columns目录视图来查看相关索引键与包含的列。

在这个系统目录视图中显示的部分选项还没有讲到，但是表5-2中描述了我已经讨论的一些选项。 204

表5-2 系统目录视图sys.indexes的子集

参 数	描 述
object_id	这个参数是索引属于的表或视图的对象标识符。可以使用OBJECT_NAME函数来显示表或视图的名称，或用OBJECT_ID将表或视图的名称转换为它的数据库对象标识符
name	这个参数表示索引的名称
index_id	当index_id = 0时，索引是堆。当index_id = 1时，索引是聚集索引。当index_id > 1时，它是非聚集索引
type	这个参数指定了索引类型，为0时是堆，1为聚集索引，2是非聚集索引，3是XML索引，4是空间
type_desc	这个参数定义了索引类型描述
is_unique	当is_unique = 1时，索引是唯一索引
is_primary_key	当is_primary_key = 1时，索引是主键约束的结果
is_unique_constraint	当is_unique_constraint = 1时，索引是唯一约束的结果

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

5.1.6　禁用索引

在这个技巧中，将展示如何在SQL Server查询中禁用索引。禁用索引在SQL Server中保留了元数据定义，但会导致索引不可用。建议将禁用索引作为索引故障处理技术，或如果发生磁盘错误并且你要延迟索引重建时再使用。

警告　如果禁用聚集索引，要牢记表索引数据将不再是可访问的。这是因为聚集索引的叶级别就是实际的表数据本身。并且，重新启用索引意味着重新创建或重组织索引（更多信息请看“解析”部分）。

通过使用ALTER INDEX命令禁用索引。语法如下：

```
ALTER INDEX index_name ON
table_or_view_name DISABLE
```

该命令有两个参数：索引的名称和创建的索引所在的表或视图的名称。

在这个示例中，禁用了表TerminationReason的UNI_TerminationReason索引：

```
ALTER INDEX UNI_TerminationReason ON
HumanResources.TerminationReason DISABLE
```

解析

这个技巧演示了如何禁用索引。如果索引是被禁用的，索引的定义还保留在系统表中，但用户不
205 能再使用索引。对于表上的非聚集索引，索引数据真正从数据库中删除了。对于表上的聚集索引，数据仍留在磁盘上，但因为索引是禁用的，你不可以查询它。对于视图上的聚集或非聚集索引，索引数据从数据库中被删除。

要重新启用索引，可以使用带有DROP_EXISTING命令（本章稍后会介绍）的CREATE INDEX或使用ALTER INDEX REBUILD（第23章中会讨论）。重建禁用的非聚集索引将重用原始索引使用的现有空间。

5.1.7　删除索引

在这个技巧中，将展示如何从表或视图中删除索引。当你删除索引时，它是从数据库中被物理删除的。如果它是聚集索引，表数据将以无序的（堆）形态保留。可以使用DROP INDEX命令从数据库中完全地删除索引。基本的语法如下：

```
DROP INDEX <table_or_view_name>.<index_name> [ ,...n ]
```

这个示例演示了从表中删除一个索引：

```
DROP INDEX HumanResources.TerminationReason.UNI_TerminationReason
```

解析

可以使用DROP INDEX命令从表中删除一个或多个索引。删除索引将释放索引所占的空间，并且从数据库中删除索引的定义。不可以使用DROP INDEX删除因创建PRIMARY KEY或UNIQUE CONSTRAINT而产生的索引。如果你删除在其上拥有非聚集索引的聚集索引，为了交换聚集索引键到堆的行标识符，那些非聚集索引也将被重建。

5.1.8　使用 DROP_EXISTING 改变既有索引

在这个技巧中，将展示如何在单个执行过程中删除和重建索引，以及如何改变既有索引的键列定义。ALTER INDEX可以用来改变索引选项、重建或重组织索引（在第23章中有讲）以及禁用索引，但不

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

可以用来实际地添加、删除或重排索引中的列。

但是，可以使用CREATE INDEX...DROP_EXISTING来改变既有索引的列定义。这个选项还拥有在单条命令中删除和重建索引的优点（取代同时使用DROP INDEX和CREATE INDEX）。而且，在聚集索引上使用DROP_EXISTING也不会引起既有的非聚集索引自动重建，除非索引列定义已发生改变。

第一个示例演示了重建非聚集索引（未改变列定义）：

```
CREATE NONCLUSTERED INDEX NCI_TerminationReason_DepartmentID ON
HumanResources.TerminationReason
(DepartmentID ASC)
WITH (DROP_EXISTING = ON)
GO
```

下一步，将新的列添加到既有的非聚集索引中：

```
CREATE NONCLUSTERED INDEX NCI_TerminationReason_DepartmentID ON
HumanResources.TerminationReason
(ViolationSeverityLevel, DepartmentID DESC)
WITH (DROP_EXISTING = ON)
GO
```


解析

在第一个示例中，CREATE INDEX没有改变既有索引定义的任何内容，只是通过使用DROP_EXISTING子句重建了索引。重建索引可以帮助对数据的碎片进行整理，这些内容在第23章会详细讲解。

在第二个语句中，将一个新列添加到既有的索引中，并且把它放置在挨着DepartmentID的前面。使用新索引建列重建索引，将它变为组合索引。

但是你不可以使用DROP_EXISTING来改变索引的名称。要完成这个操作，需要使用DROP INDEX和带有新索引名称的CREATE INDEX。

5.2 控制索引创建的性能和并发性

迄今为止，本章已经介绍了如何定义索引，但注意你也可以决定在什么情形下才创建索引。例如，当在SQL Server中创建索引时，为了提升性能，可以指定使用并行执行计划，比如使用多处理器来帮助完成耗时的索引创建。除此之外，也可以指示SQL Server在tempdb中创建索引，这样不会导致索引所在的数据库中的文件增长操作。如果使用企业版，现在也可以在使用新的ONLINE选项创建的索引过程中允许并发用户查询访问底层的表。

随后的3个技巧将演示提升索引创建性能和提升操作过程中用户并发性的方法。

5.2.1 在 tempdb 中创建临时索引

在这个技巧中，将展示如何把索引创建放在系统数据库tempdb中完成。系统数据库tempdb用来管理用户连接、临时表、临时存储过程或SQL Server实例中处理查询需要的临时工作表。根据SQL Server实例中数据库的活动性，有时可以通过将tempdb数据库从其他数据库分离出来，使用它自己的磁盘阵列来获得性能方面的提升。如果索引创建时间比期望的要长很多，可以尝试使用索引选项SORT_IN_TEMPDB来提升索引创建性能（对于大型表）。这个选项把临时索引创建结果放置在tempdb数据库中，而不是使用索引所在的用户数据库。

这个选项可以用在CREATE INDEX和ALTER INDEX中，它的语法如下所示：

```
WITH (SORT_IN_TEMPDB = { ON | OFF })
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个选项的默认值为OFF。在这个示例中，启用这个选项来创建新的非聚集索引：

```
CREATE NONCLUSTERED INDEX NI_Address_PostalCode ON
Person.Address (PostalCode)
WITH (SORT_IN_TEMPDB = ON)
```

解析

207 SORT_IN_TEMPDB选项启用了使用tempdb数据库保存中间的索引结果。这个选项可以减少创建大型表索引所花费的时间，但作为代价，系统数据库tempdb将需要额外的空间来参与这个操作。

5.2.2　控制索引创建的并行执行计划

在这个技巧中，将展示如何控制用来处理单个查询的处理器的数量。如果使用SQL Server企业版的多处理器服务器，可以通过使用MAXDOP索引选项来控制/限制在索引创建操作中潜在使用的处理器的数量。在这里，并行（即使用两个或更多处理器完成一个查询语句）可以潜在地提升索引创建操作的性能。

这个选项可以用在CREATE INDEX和ALTER INDEX中，其语法如下所示：

```
MAXDOP = max_degree_of_parallelism
```

这个选项的默认值为0，意思是SQL Server在操作中可以选择任一或所有可用的处理器。MAXDOP值为1则禁用索引创建的并发性。

提示　限制索引创建的并发性可能会提升在创建过程中用户活动的并发性，但也可能增加索引创建花费的时间。

这个示例演示了如何控制在索引创建过程中并行执行计划使用的处理器的数量：

```
CREATE NONCLUSTERED INDEX NI_Address_AddressLine1 ON
Person.Address (AddressLine1)
WITH (MAXDOP = 4)
```

解析

在这个技巧中，索引创建被限制为4个处理器：

```
WITH (MAXDOP = 4)
```

设置MAXDOP并不能保证SQL Server将实际使用指定的处理器的数量。它只是确保SQL Server不会超过MAXDOP的限定值。

5.2.3　在索引创建过程中允许用户表访问

在这个技巧中，将展示如何在索引创建时允许查询活动继续访问索引。如果使用SQL Server企业版，可以通过使用新的ONLINE选项允许并发用户查询在索引创建过程中访问底层的表。下面的技巧演示了这个内容：

```
CREATE NONCLUSTERED INDEX NCI_ProductVendor_MinOrderQty ON
Purchasing.ProductVendor(MinOrderQty)
WITH (ONLINE = ON)
```

208

解析

在创建索引的WITH子句中使用新的ONLINE选项，在索引创建过程中不会产生长时间的表锁。这可以为包含频繁访问数据的大型索引提供更好的并发性。当ONLINE选项设置为ON时，在索引创建过程中

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

只有共享意向锁保持在源表上，而不是索引创建过程中的默认的长时间表锁保持行为。

5.3 索引选项

下面的3个技巧讲到了影响性能的几个选项，每个选项的影响方式不同。例如新关键字INCLUDE允许你添加非键列到非聚集索引中。这允许你创建覆盖的索引，它可以不访问聚集索引数据就将数据返回给用户。

第二个技巧将讨论PAD_INDEX与FILLFACTOR选项怎样决定如何设置填充索引叶级别页和索引中间级别的初始的行百分比。技巧将讨论填充因子是如何影响查询的性能和插入、更新以及删除操作的性能的。

第三个技巧将讲解如何禁用特定索引的某些锁定类型。正如将在技巧中讨论到的，使用这些选项允许你在查询访问索引时控制并发性与资源使用。

5.3.1 使用索引 INCLUDE

在这个技巧中，将展示如何在非聚集索引中包含非键列。*覆盖查询*（covering query）是指其引用的所有列均能在非聚集索引中找到的查询。这个方案经常会带来出众的查询性能，因为SQL Server不需要从聚集索引或堆中检索实际的数据——它只需要读取在非聚集索引中存储的数据。但是它的缺点是最多只能包含16列或900字节的索引键。

这个问题的一个解决方案是新引入的关键字INCLUDE，它允许添加最多1023个非键列到非聚集索引，通过创建覆盖索引帮助提升查询性能。这些非键列没有存储在索引的所有级别上，而只是存在于非聚集索引的叶级别上。

使用INCLUDE的CREATE NONCLUSTERED INDEX的语法如下：

```
CREATE NONCLUSTERED INDEX index_name
    ON table_or_view_name ( column [ ASC | DESC ] [ ,...n ] )
INCLUDE ( column [ ,... n ] )
```

第一个列列表是键索引列，在INCLUDE之后的列列表是非键列。在这个示例中，将新的大型对象数据类型列添加到TerminationReason表中。删除并重建在DepartmentID上的既有索引，这次添加新的非键值到索引中：

```
ALTER TABLE HumanResources.TerminationReason
ADD LegalDescription varchar(max)

DROP INDEX
HumanResources.TerminationReason.NI_TerminationReason_TerminationReason_DepartmentID

CREATE NONCLUSTERED INDEX NI_TerminationReason_TerminationReason_DepartmentID
ON HumanResources.TerminationReason (TerminationReason, DepartmentID)
INCLUDE (LegalDescription)
```

209

解析

这个技巧演示了增强非聚集索引有效性的新技术。示例从创建新的varchar(max)数据类型列开始。因为它的数据类型，不可以在索引中将它当作键值来使用，但在关键字INCLUDE中使用它将允许你引用新的大型对象数据类型。然后使用INCLUDE和新的非键列删除并重建表TerminationReason上的既有索引。

只可以对非聚集索引使用INCLUDE（在覆盖查询上派得上用场），并且仍然不可以包含废弃的image、ntext以及text数据类型。如果额外的非键值使索引大小增长得太显著，你可能会损失一些覆盖查询可

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

以给予的查询性能，因此要确认测试比较之前和之后的性能。

5.3.2 使用 PAD_INDEX 和 FILLFACTOR

在这个技巧中，将展示如何设置填充索引叶级别页以及索引中间级别的初始的行百分比。索引的填充因子百分比是指首次创建索引时索引页的叶级别充满程度。如果没有显式设置填充因子，则它默认为0，即尽最大可能来填充页（SQL Server会保留一些可用空间——对于单个索引行足够了）。然而保留一些可用空间允许插入新行时不拆分页。添加新行到充满的索引页会引发页拆分。为了得到空间，会从既有的充满的页移动一半的行到新页。大量的页拆分会减慢INSERT操作。但是另一方面，充满的数据页允许更快的读取活动，因为数据库引擎可以从更少的数据页中检索更多的行。

PAD_INDEX选项只用来连接FILLFACTOR，它指定在索引的中间页级别页打开的自由空间特定的百分比。

这些选项在CREATE INDEX和ALTER INDEX命令的WITH子句中设置。语法如下：

```
WITH (PAD_INDEX = { ON | OFF }
  | FILLFACTOR = fillfactor)
```

在这个示例中，使用填充因子50%并启用PAD_INDEX来删除并重建索引：

```
DROP INDEX
HumanResources.TerminationReason.NI_TerminationReason_TerminationReason_DepartmentID

CREATE NONCLUSTERED INDEX NI_TerminationReason_TerminationReason_DepartmentID
ON HumanResources.TerminationReason
(TerminationReason ASC, DepartmentID ASC)
WITH (PAD_INDEX=ON, FILLFACTOR=50)
```

解析

在这个技巧中，填充因子配置为50%，为新行保留50%的索引页自由空间。同时也启用了PAD_INDEX，因此中间索引页也将保留一半的自由空间。两个选项都用在了CREATE INDEX中的WITH子句中：

```
WITH (PAD_INDEX=ON, FILLFACTOR=50)
```

使用FILLFACTOR是读与写之间的一个平衡操作。例如，100%的填充因子可以提升读取的性能，但是会减慢写活动的性能，引发频繁的页拆分，因为数据库引擎为了在数据页中得到空间必须持续地交换行的位置。太低的填充因子会给行插入带来益处，但它也会减慢读取操作，因为要检索所有需要的行必须访问更多的数据页。经验做法是，为几乎没有数据修改活动的表使用100%填充因子，低活动的使用80%～90%，中等活动的使用60%～70%，为索引键上的高活动使用50%或更低的百分比。

210

5.3.3 禁用页和/或行索引锁定

在这个技巧中，我将展示如何改变被锁定的特定索引的锁定资源类型。第3章已经讨论了SQL Server中的各种锁定类型和资源。具体地说，SQL Server可以锁定各种资源从小（行与键锁）到中（页锁、区）到大（表、数据库）。假定大量的查询同时从同一表和相关索引请求数据，多个细粒度的锁有助于提高查询并发性。然而大量的锁会占用内存，降低SQL Server实例的整体性能。另一方面，粗粒度的锁增加了内存资源的可用性，但同时也减少了查询的并发性。

你可以创建一个当索引被查询时限制某些锁定类型的索引。具体地说，你可以指定是否允许页锁或行锁。

在通常情况下，应该让SQL Server自动决定何种锁定类型是最佳的。然而，也会出现你希望临时

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

限定某些资源锁定类型的情况，比如说故障诊断或存在严重的性能问题时。

为CREATE INDEX和ALTER INDEX配置这些选项的语法如下：

```
WITH ( ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF })
```

这个技巧展示了如何禁用数据库引擎在索引上放置行或页锁的能力，强制它使用表锁：

```
-- 禁用页锁，表锁和行锁仍可以使用
CREATE INDEX NI_EmployeePayHistory_Rate ON
HumanResources.EmployeePayHistory (Rate)
WITH (ALLOW_PAGE_LOCKS=OFF)

-- 禁用页锁和行锁，只可以使用表锁
ALTER INDEX NI_EmployeePayHistory_Rate ON
HumanResources.EmployeePayHistory
SET (ALLOW_PAGE_LOCKS=OFF,ALLOW_ROW_LOCKS=OFF )

-- 允许页锁和行锁
ALTER INDEX NI_EmployeePayHistory_Rate ON
HumanResources.EmployeePayHistory
SET (ALLOW_PAGE_LOCKS=ON,ALLOW_ROW_LOCKS=ON )
```

解析

这个技巧演示了3个变化。第一个查询在表上创建了新的索引，配置数据库引擎不可以在索引上实行页锁。

```
WITH (ALLOW_PAGE_LOCKS=OFF)
```

在下一语句中，将页锁和行锁设为OFF（索引中这些选项的默认值为ON）：

```
ALTER INDEX NI_EmployeePayHistory_Rate ON
HumanResources.EmployeePayHistory
SET (ALLOW_PAGE_LOCKS=OFF,ALLOW_ROW_LOCKS=OFF )
```

在最后的语句中，重新启用页锁和行锁：

```
SET (ALLOW_PAGE_LOCKS=ON,ALLOW_ROW_LOCKS=ON )
```

只有当你有合理的理由这样做时才去删除锁定选项——例如可能拥有产生太多行锁的活动，它可以用光内存资源。你可能希望SQL Server使用粗粒度的页锁或表锁来取代行锁。

5.4 管理超大型索引

本章的最后两个技巧讲了管理超大型索引的方法，但是这里演示的特性也适用于小型和中型索引。例如你可以指定在另一个文件组中创建索引。这样做具有更好的可管理性与性能，因为后面你可以通过文件组实施不同的备份。如果文件组的文件存在于不同的磁盘阵列上，也会提升查询的I/O性能。

正如在第4章开始时讨论的，你也可以实施索引分区。分区允许你将索引数据集拆分为更小的数据子集。正如将在以下技巧中讨论到的，如果大型索引被分割到不同的分区上，它一定可以提升查询的性能（特别是对于超大型索引来说）。

SQL Server 2008也引入了筛选索引特性和在页级和行级上压缩数据的能力。筛选索引特性允许你为数据子集创建索引和相关统计信息。例如，如果输入的查询只命中列中小部分的值，你可以创建只关注那些普遍值的筛选索引——和完整表索引相比，这样可以减少总的索引的大小，也可以提高底层统计信息的精确性。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

至于新的压缩特性，在企业版和开发版都是可用的，你现在可以为索引或特定分区指定行压缩或页压缩。我最初在第4章中的CREATE TABLE和ALTER TABLE中演示过这个特性。在本章中，我将继续讨论如何使用CREATE INDEX和ALTER INDEX来启用压缩。

5.4.1 在文件组上创建索引

在这个技巧中，将展示如何在特定的文件组上创建索引。如果没有显式地指定，索引会创建在与底层表相同的文件组中。在CREATE INDEX命令中使用ON子句来实现它：

```
ON filegroup_name | default
```

这个选项使用显式的文件组名称或数据库默认文件组（文件组的更多信息请看第22章）。

这个示例演示了如何显式地定义索引存储在哪一个文件组中。首先，在数据库Adventureworks中创建新的文件组：

```
ALTER DATABASE AdventureWorks
ADD FILEGROUP FG2
```

下一步，将新文件添加到数据及新创建的文件组中：

```
ALTER DATABASE AdventureWorks
ADD FILE
(   NAME = AW2,
    FILENAME = 'c:\Apress\aw2.ndf',
    SIZE = 1MB
)
TO FILEGROUP FG2
```

212

最后，创建新索引，指定它存储在新创建的文件组中：

```
CREATE INDEX NI_ProductPhoto_ThumnailPhotoFileName ON
Production.ProductPhoto (ThumbnailPhotoFileName)
ON [FG2]
```

解析

在技巧的第一部分，在数据库AdventureWorks中使用ALTER DATABASE命令创建了名称为FG2的新文件组。之后在新文件组中创建了新的数据库数据文件。最后，将新索引创建在FG2文件组中。ON子句在方括号中指定了索引所在的文件组名称：

```
ON [FG2]
```

文件组可以用来帮助管理超大型数据库，通过文件组--方面可以进行单独的备份，另一方面如果文件组的文件存在于分离的磁盘阵列中时也会提升I/O性能。

5.4.2 实现索引分区

在这个技巧中，将展示如何为非聚集索引进行分区。在第4章中，我演示了表分区。分区可以为大型表带来易管理性、可扩展性以及性能提升，这是因为分区允许将数据集分解成更小的数据子集。依赖于索引键，表上的索引也可以是非常大的。应用索引分区的概念，如果大型索引拆分到不同的分区上，则一定会提升查询的性能。如果查询的目标数据只是存在于一个分区上，将会带来益处，因为SQL Server只需锁定选定的分区，而不是访问索引的所有分区。

这个技巧现在将演示使用第4章中在表Sales.WebSiteHits上创建的HitDateRangeScheme分区方案进行索引分区：

```
CREATE NONCLUSTERED INDEX NI_WebSiteHits_WebSitePage ON
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Sales.WebSiteHits (WebSitePage)
ON [HitDateRangeScheme] (HitDate)
```

解析

使用ON子句应用分区方案。

```
ON [HitDateRangeScheme] (HitDate)
```

注意，尽管HitDate列不是非聚集索引键，但是它包含在分区方案中，匹配表的索引键。当索引和表使用同一分区方案时，它们被称为“对齐”。

你可以选择为索引使用与表不同的分区方案，但是必须使用同一数据类型参数、分区的数量和边界值。非对齐的索引可以在合并联结中获得益处——意思是如果两个表中的两列频繁联结，并且使用同一分区函数、同一数据类型、同样的分区数与边界，则可能会提升查询联结性能。然而，出于管理与性能方面的原因，通常的方法将更可能是使用索引与表的对齐分区方案。

213

5.4.3 为行的子集做索引

SQL Server 2008引入了创建筛选非聚集索引的能力，以支持仅请求少量表行的查询。现在可以使用包含筛选谓词的CREATE INDEX命令通过只对满足特定条件的行进行索引来减少索引大小。减少索引大小节省了磁盘空间，并潜在地提升了查询性能，因为只需要读取一小部分必须处理的索引条目。

筛选谓词允许使用几个比较运算符，包括IS、IS NOT、=、<>、>、<等。在这个技巧中，我将演示如何为AdventureWorks数据库中的一个比较大的表Sales.SalesOrderDetail添加筛选索引。为了开始这个示例，假设有一个对UnitPrice列操作的普通查询：

```
SELECT SalesOrderID
FROM Sales.SalesOrderDetail
WHERE UnitPrice BETWEEN 150.00 AND 175.00
```

然后再假设执行这个查询的人是唯一一个在搜索谓词中使用UnitPrice列的人，并且当他执行这个查询时，只关心$150到$175之间的值。为这个列创建的完整的索引可能会被认为是浪费。如果这个查询经常被执行，并且每次都对基础表实施全聚集索引扫描，这可能会引起性能问题。

我已经描述了在UnitPrice列上筛选索引的理想情况。你可以按如下方式创建筛选索引：

```
CREATE NONCLUSTERED INDEX NCI_UnitPrice_SalesOrderDetail
ON Sales.SalesOrderDetail(UnitPrice)
WHERE UnitPrice >= 150.00 AND UnitPrice <= 175.00
```

搜索UnitPrice并且在定义的筛选谓词范围内搜索的查询很可能使用筛选索引，而不是实施全索引扫描或使用全表索引。

在第二个示例中，假定以两个不同的ID查询产品。在这个示例中，我还查询所有订单数量大于10的记录，但是，这不是希望的筛选情景——仅仅是产品ID的筛选：

```
SELECT SalesOrderDetailID
FROM Sales.SalesOrderDetail
WHERE ProductID IN (776, 777) AND
      OrderQty > 10
```

这个查询实施了聚集索引扫描。可以通过增加筛选索引来提升性能，这样查询就会使用非聚集索引进行索引查找，而不是使用聚集索引扫描。下面是创建筛选索引的方法：

```
CREATE NONCLUSTERED INDEX NCI_ProductID_SalesOrderDetail
ON Sales.SalesOrderDetail(ProductID,OrderQty)
WHERE ProductID IN (776, 777)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

结果使用的I/O较少，因为查询可以以更小的筛选索引进行操作。

解析

这个技巧演示了如何使用筛选索引特性来创建需要比全表索引更少存储空间的微调的索引。筛选
214 索引需要你理解传入的对数据库中表的查询的性质。如果大部分查询都是查询表中一小部分数据，筛选索引将会提升I/O性能，也会减少磁盘存储。

CREATE INDEX语句与它的原始形式相比没有太大的改变。在ON子句之后使用WHERE子句来实现筛选（如果使用INCLUDE，WHERE需跟随其后）：

```
CREATE NONCLUSTERED INDEX NCI_UnitPrice_SalesOrderDetail
ON Sales.SalesOrderDetail(UnitPrice)
WHERE UnitPrice >= 150.00 AND UnitPrice <= 175.00
```

筛选谓词允许使用IN、IS、IS NOT、=、<>、>、>=、!>、<、<=和!<等运算符来完成一些简单的逻辑。你也要了解随筛选索引一起创建的筛选统计信息。这些统计信息使用相同的筛选谓词，并且由于从更小的行集中采样，统计信息的结果更加精确。

5.4.4　减小索引的大小

正如在第4章提到的，SQL Server 2008企业版和开发版引入了对表、索引和相应分区的页级和行级压缩。第4章演示了如何与CREATE TABLE和ALTER TABLE命令一起使用DATA_COMPRESSION子句来启用压缩。还讨论了如何压缩聚集索引和堆。对于非聚集索引，使用CREATE INDEX和ALTER INDEX来实现压缩。语法是相同的，指定了值为NONE、ROW或PAGE的DATA_COMPRESSION选项。随后的示例演示了对非聚集索引增加PAGE级压缩（以第4章中创建的示例表ArchiveJobPosting为基础）：

```
CREATE NONCLUSTERED INDEX NCI_SalesOrderDetail_CarrierTrackingNumber
ON Sales.SalesOrderDetail (CarrierTrackingNumber)
WITH (DATA_COMPRESSION = PAGE)
```

可以在创建后使用ALTER TABLE来修改压缩级别。在这个示例中，使用ALTER INDEX将压缩级别更改为行级压缩：

```
ALTER INDEX NCI_SalesOrderDetail_CarrierTrackingNumber
ON Sales.SalesOrderDetail
REBUILD
WITH (DATA_COMPRESSION = ROW)
```

解析

这个技巧演示了为非聚集索引启用行压缩或页压缩。添加压缩的过程几乎与为聚集索引或堆添加压缩的过程一模一样，都使用DATA_COMPRESSION索引选项。当创建新的索引时，WITH子句跟在索引键
215 的定义之后。当修改既有索引时，WITH子句在REBUILD关键字之后。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第6章 全文搜索



全文搜索功能提供了基于字符和二进制数据的智能单词（和短语）搜索，使用全文搜索的执行效率会比普通的LIKE查询好很多。

现在在SQL Server 2008中，全文搜索功能集成到数据库中了。全文目录不再单独存储在文件系统中，现在也集成到了数据库本身当中。全文索引和查询支持功能也不再依赖于SQL Server早期版本中的分离的MSFTESQL服务。

注解 SQL Server 2008也在数据库中全面集成非索引字（以前称为干扰词），允许创建自己的非索引字表和相关非索引字。早期版本在数据库外部使用干扰词文件。我将在6.1.6节中讨论这个功能。

在本章的第一部分，将介绍如何在数据库中使用T-SQL启用全文搜索能力。在本章的第二部分，将演示如何使用基础和高级T-SQL谓词查询全文索引。

6.1 全文索引和全文目录

全文索引（full-text index）能让我们更灵活地搜索非结构化的文本数据，并且提供了比LIKE查询好很多的性能。与普通的B树聚集索引或非聚集索引不同的是，全文索引是由文本数据的索引标记组成的压缩的索引结构。标记是SQL Server在索引过程中标识的单词或字符串。使用特殊的全文搜索功能可以在字符模式之上扩展单词或短语搜索，并基于其他单词的变形词、同义词、通配符和近义词进行搜索。

全文目录可以不包含全文索引，也可以包含数量不等的全文索引，从SQL Server 2008开始，全文目录存储在数据库中。（在早期的版本中，它存储在SQL Server实例服务器的本地硬盘上。）全文目录可包含用于在单个数据库内索引一个或多个表的全文索引。

SQL Server使用几个T-SQL命令来创建、修改和移除全文目录和全文索引对象，在随后的一组技巧中将会演示。

6.1.1 创建全文目录

创建全文目录的最简单形式是只提供全文目录的名字。然而还有其他一些选项，CREATE FULLTEXT CATALOG的扩展语法如下：


```
CREATE FULLTEXT CATALOG catalog_name
    [ON FILEGROUP 'filegroup']
    [IN PATH 'rootpath']
    [WITH ACCENT_SENSITIVITY = {ON|OFF}]
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
[AS DEFAULT]
[AUTHORIZATION owner_name ]
```

表6-1描述了此命令的参数。

表6-1　CREATE FULLTEXT CATALOG参数

参　数	描　述
catalog_name	这个选项指定了新的全文目录的名称
filegroup	这个参数指定全文目录存放的文件组。如果未指定，将使用数据库默认的文件组
rootpath	这是SQL Server 2008中废弃的选项，已不再使用
ACCENT_SENSITIVITY={ON\|OFF}	此选项用于选择在此全文目录中创建全文索引是否区分重音，ACCENT_SENSITIVITY定义了SQL Server是否区分重音字符和非重音字符
AS DEFAULT	此选项将该全文目录定义为默认全文目录，在数据库中创建的、未显式指定全文目录的全文索引将使用该全文目录
owner_name	AUTHORIZATION选项决定了新的全文目录的所有者，可以选择一个数据库用户或者是角色作为owner_name

在下面第一个例子中，在AdventureWorks数据库中创建了一个新的全文目录（注意，一个全文目录只能属于一个数据库）：

```
USE AdventureWorks
GO
CREATE FULLTEXT CATALOG cat_Production_Document
```

在第二个例子中，建立一个区分重音的全文目录：

```
USE AdventureWorks
GO
CREATE FULLTEXT CATALOG cat_Production_Document_EX2
WITH ACCENT_SENSITIVITY = ON
```

解析

在这个技巧中，演示了如何使用CREATE FULLTEXT CATALOG命令创建一个全文目录。这个命令创建了一个可以用来聚合一个或更多全文索引的实例逻辑实体。

218　全文目录创建后，可以在它下面创建全文索引，这些内容将在下面的技巧中阐述。

6.1.2　创建全文索引

在这个技巧中，演示如何在表的列上创建全文索引。然后我们就能利用本章稍后会讲述的更灵活的搜索能力。

创建全文索引的命令是CREATE FULLTEXT INDEX，部分语法如下所示：

```
CREATE FULLTEXT INDEX ON table_name
      [ ( { column_name
              [ TYPE COLUMN type_column_name ]
              [ LANGUAGE language_term ]
          } [ ,...n]
              ) ]
    KEY INDEX index_name
        [ ON fulltext_catalog_name]
        [ WITH [ ( ] <with_option> [ ,...n] [ ) ] ]
[;]
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<with_option>::=
  {
   CHANGE_TRACKING [ = ] { MANUAL | AUTO | OFF [, NO POPULATION ] }
  | STOPLIST [ = ] { OFF | SYSTEM | stoplist_name }
  }
```

表6-2详细描述了此命令的参数。

表6-2 CREATE FULLTEXT INDEX参数

参 数	描 述
table_name	这个参数指定了要创建全文索引的表的名称，一个表只允许有一个全文索引
column_name	这个参数定义了要建立全文索引的一列或多列的名字，数据类型只能为varchar、nvarchar、char、nchar、xml、varbinary、text、ntext、image
type_column_name	TYPE COLUMN关键字标记用来指定表中的列，告诉全文索引在varbinary(max)或image数据类型的列中保存了什么类型的数据。SQL Server可以解释不同的文件类型，但是它必须准确地知道如何去处理
language_term	LANGUAGE为可选关键字，也可以用在列列表中用来指出列中数据的语言。指出语言能帮助SQL Server确定在全文索引处理过程中如何进行数据解析及如何进行语言解释。可以在sys.fulltext_languages表中查询到有哪些可使用的语言
index_name	为了能在某个表上建立全文索引，表必须包含一个唯一的单键非空列。例如，一个单列主键或一个非空的且具有UNIQUE约束的列。CREATE FULLTEXT INDEX命令的KEY INDEX子句标识指定表上所需的唯一键列
fulltext_catalog_name	ON子句指定全文索引要存储的全文目录。如果在全文索引创建前设定过默认全文目录，而且没有使用该选项，那么该全文索引将创建在默认的全文目录中。但是，如果没有定义默认全文目录，全文索引的创建将会失败
CHANGE_TRACKING {MANUAL \| AUTO \| OFF [, NO POPULATION]}	这个参数决定全文服务如何监测到用户数据的变化。基于这个设置，全文索引可以随着表数据的更新而自动更新。也可以选择在某个所选择的时间点或者计划表手动更新全文索引。指定AUTO选项后，全文索引会随着表数据的修改而自动更新。MANUAL选项意味着，全文索引要通过用户手动传播或是通过SQL Server代理按计划传播。OFF选项意味着，SQL Server不保留对索引数据的更改。声明含NO POPULATION的OFF选项意味着，全文索引建立之后不会被立即填充。在使用这个选项时，全文索引只有在用户执行ALTER FULLTEXT INDEX后才会被填充。这个内容会在下个技巧中阐述
STOPLIST [=]{OFF \| SYSTEM \| stoplist_name}	非索引字表包含非索引字的列表，它是可以被搜索忽略的字符串。默认选项是SYSTEM，意思是使用默认系统非索引字表。当这个选项设为OFF时，不使用非索引字表。否则，指定stoplist_name来使用用户定义的非索引字表

在这个技巧的示例中，我们在AdventureWorks数据库的Production.Document表上建立了一个新的全文索引（我们将在后面的技巧中讲到如何查询这个全文索引）。全文索引的列是DocumentSummary，FileExtension是包含了指向列的文档类型的指针的列：

```
USE AdventureWorks
GO

CREATE FULLTEXT INDEX ON Production.Document
(DocumentSummary, Document TYPE COLUMN FileExtension)
KEY INDEX PK_Document_DocumentNode
ON cat_Production_Document
WITH CHANGE_TRACKING AUTO,
     STOPLIST = SYSTEM
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个技巧中，我们在Production.Document表的DocumentSummary列（该列数据类型为varchar(max)）和Document列（该列数据类型为varbinary(max)）上创建了一个全文索引。在这段代码中，第一行指定了全文索引基于哪一个表：

220
```
CREATE FULLTEXT INDEX ON Production.Document
```

第二行代码指定了要全文索引的列，然后列类型指针说明了列的文档类型。在这个例子中，对DocumentSummary和Document列做了索引。由于Document是varbinary(max)数据类型，指定了在Document列中包含文件类型的列：

```
(DocumentSummary, Document TYPE COLUMN FileExtension)
```

要记住，只有要进行全文索引的列类型是varchar(max)或image时，才有必要使用TYPE COLUMN子句来帮助SQL Server解释存储的数据，普通的文本数据类型（如char、varchar、nchar、nvarchar、text、ntext和xml）不需要使用TYPE COLUMN子句。

下一步是标识键（表的非空唯一列）的名称：

```
KEY INDEX PK_Document_DocumentNode
```

ON子句指定全文索引要存储的全文目录：

```
ON cat_Production_Document
```

下一步，指定全文索引的填充方式：

```
WITH CHANGE_TRACKING AUTO
```

最后，指定STOPLIST选项——使用系统默认非索引字表：

```
STOPLIST = SYSTEM
```

全文索引建立后，就可以查询它了。但是在查询它之前，还需要知道其他修改或删除全文索引和全文目录的命令。

6.1.3 修改全文目录

在这个技巧中，会演示ALTER FULLTEXT CATALOG命令，可以用它来实现以下几种操作。

- 修改区分重音设置，区分重音定义了SQL Server在查询过程中区分重音及非重音字符还是同等对待这两类字符。
- 设置全文目录为默认的数据库目录。
- REBUILD（重新建立）其中有全部的索引的全文目录。
- REORGANIZE（重组织）全文目录，这个操作可以优化内部索引和目录全文结构。这个过程被称为主合并，意思是将那些细小的分散的全文索引物理（不是逻辑地）处理到一个大的完整的全文索引中，以提升性能。

下面是ALTER FULLTEXT CATALOG的语法：

```
ALTER FULLTEXT CATALOG catalog_name
{ REBUILD [WITH ACCENT_SENSITIVITY = {ON|OFF} ]
     | REORGANIZE
     | AS DEFAULT
}
```

221

表6-3详细描述了此命令的参数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表6-3 ALTER FULLTEXT CATALOG参数

参 数	描 述
REBUILD	REBUILD选项会重生成目录
[WITH ACCENT_SENSITIVITY = {ON\|OFF}]	ACCENT_SENSITIVITY选项必须要和REBUILD选项同时使用
REORGANIZE	这个选项可以使SQL Server优化目录结构和内部索引
AS DEFAULT	这个选项将全文目录设置为默认的数据库目录

在本技巧的第一个例子中，用REORGANIZE关键字来优化全文目录：

```
ALTER FULLTEXT CATALOG cat_Production_Document
REORGANIZE
```

第二个例子中，将全文目录设置为数据库的默认全文目录：

```
ALTER FULLTEXT CATALOG cat_Production_Document
AS DEFAULT
```

在下面的例子中，重建了一个禁用区分重音的全文目录（和其中的全部索引）。

```
ALTER FULLTEXT CATALOG cat_Production_Document
REBUILD WITH ACCENT_SENSITIVITY = OFF
```

解析

在这个技巧中，用ALTER FULLTEXT CATALOG优化全文索引及内部数据结构，将全文目录设置为数据库默认的全文目录，并重建全文目录和其中的索引。这个命令用来管理已建立的全文目录，保证索引后的表在数据修改时能有最佳的性能。

6.1.4 修改全文索引

ALTER FULLTEXT INDEX命令用来改变全文索引的属性，也可以控制/启动全文索引的填充。语法如下：

```
ALTER FULLTEXT INDEX ON table_name
   { ENABLE
   | DISABLE
   | SET CHANGE_TRACKING { MANUAL | AUTO | OFF }
   | ADD ( column_name
     [ TYPE COLUMN type_column_name ]
     [ LANGUAGE language_term ] [,...n] )
     [ WITH NO POPULATION ]
   | DROP ( column_name [,...n] )
     [WITH NO POPULATION ]
   | START { FULL | INCREMENTAL | UPDATE } POPULATION
   | {STOP | PAUSE | RESUME } POPULATION
   | SET STOPLIST { OFF| SYSTEM | stoplist_name }
     [WITH NO POPULATION] }
```

表6-4详细描述了此命令的参数。

222

表6-4 ALTER FULLTEXT INDEX参数

参 数	描 述
table_name	这个参数指定了要修改的全文索引所在的表名
ENABLE \| DISABLE	ENABLE选项激活全文索引功能。DISABLE选项关闭全文索引功能。关闭全文索引功能意味着以后对表字段的任何修改都不会被跟踪并体现在全文索引上（即使DISABLE了，全文索引查询条件仍然有效）

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

参 数	描 述
SET CHANGE TRACKING {MANUAL\|AUTO\|OFF}	MANUAL指定源索引数据的更改跟踪是按计划启用还是由用户手动进行。AUTO指定在修改索引列中的数据时，SQL Server 将自动更新全文索引。OFF指定关闭跟踪全文索引中的修改
ADD (column_name [, ...n])	这个参数表示添加到既有全文索引中的一个或多个列的名称
type_column_name	这个选项指定了用于指定在varbinary(max)和image数据类型列中保存的数据的全文索引文件类型的列
language_term	这个参数表示在列列表中使用的可选LANGUAGE关键字，指定了列中保存数据的语言
WITH NO POPULATION	如果指定这个参数，当添加或删除表的列后不填充全文索引
DROP (column_name [, ...n])	这个参数指定了要从既有全文索引中移除的列的名字
START {FULL\|INCREMENTAL\|UPDATE} POPULATION	这个选项指示全文索引根据FULL、INCREMENTAL和UPDATE来填充。FULL刷新表的每一行的索引。INCREMENTAL只刷新那些自上次填充后修改的行。仅当表包含一个类型为timestamp的列时，才能应用 INCREMENTAL。UPDATE刷新自上次索引更新以来所有插入、更新和删除的行的索引
{STOP \| PAUSE \| RESUME} POPULATION	对于非常大的表，全文索引填充会消耗巨大的系统资源。因此，你可能需要在这个过程中关闭填充。对于使用MANUAL或OFF更改跟踪设置创建的索引，可以使用STOP POPULATION选项。当正在进行完整填充时可以使用PAUSE和RESUME
SET STOPLIST {OFF \| SYSTEM \| stoplist_name }	指定SYSTEM意思是使用默认的系统非索引字表。这个选项设置为OFF时，不使用非索引字表。否则，指定stoplist_name使用用户定义非索引字表

223

在第一个例子中，我们将增加一个新的列到Production.Document表的全文索引中：

```
ALTER FULLTEXT INDEX ON Production.Document
ADD (Title)
```

下一步，启动一个全文索引填充操作：

```
ALTER FULLTEXT INDEX ON Production.Document
START FULL POPULATION
```

因为全文索引填充操作正在进行（在增加新的列到全文索引的操作中我们没有指定WITH NO POPULATION选项），该操作将会返回一个警告：

```
警告: 启动对表或索引视图'Production.Document'的全文索引填充的请求已被忽略，因为此表或索引视图当前已有活动的填充。
```

在下面的例子中，将关闭表的全文索引的更改跟踪：

```
ALTER FULLTEXT INDEX ON Production.Document
SET CHANGE_TRACKING OFF
```

它返回如下的警告[①]：

```
警告: 停止更改跟踪的请求已删除了表或索引视图'Production'上所有跟踪到的更改。
```

在本技巧最后的例子中，我们将Title列从全文索引中删除：

```
ALTER FULLTEXT INDEX ON Production.Document
DROP (Title)
```

① 该处是SQL Server 2008的bug，'Production'应为'Production.Document'。——译者注

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个技巧中，我们使用ALTER FULLTEXT INDEX命令来完成以下操作。

- 在既有的全文索引中增加一个新列。如果增加这个列能提升全文搜索性能，那么这个操作非常有用。
- 启动全文索引填充（在填充没有设置为自动更新的情况下可以这样做）。在超大型表中，你可能会希望手动控制全文索引填充而不用SQL Server随着时间手动填充。
- 关闭更改跟踪，这将移除发生在索引数据上的所有更改的日志。
- 从全文索引中删除一个列，例如，如果既有的一个全文索引中存在一个并不能提升性能的列，最好的做法就是把它从全文索引中删除，以节省更多的空间（即这个全文索引数据占用的空间）和系统资源（即更新这个数据SQL Server消耗的资源）。

还有其他一些ALTER FULLTEXT INDEX可完成的操作，包括通过DISABLE选项关闭已激活的全文索引，这将禁用全文索引（但这些全文索引的元数据还在系统表中）。当然，也可以通用ENABLE选项激活已关闭的全文索引。

224

6.1.5 检索全文目录和索引元数据

这个技巧展示如何使用系统目录视图在数据库中检索关于全文目录和索引的有用的信息。

sys.fulltext_catalogs系统目录视图返回当前数据库中全文目录的信息。例如：

```
SELECT name, path, is_default, is_accent_sensitivity_on
FROM sys.fulltext_catalogs
```

这个查询返回：

```
Name                          path    is_default    is_accent_sensitivity_on
cat_Production_Document       NULL    1             0
cat_Production_Document_EX2   NULL    0             1
```

sys.fulltext_indexs系统目录视图列出了数据库中所有的全文索引。例如：

```
SELECT object_name(object_id) table_name,
       change_tracking_state_desc, stoplist_id
FROM sys.fulltext_indexes
```

这个查询返回：

```
table_name   change_tracking_state_desc    stoplist_id
Document     OFF                          0
```

sys.fulltext_index_columns系统目录视图列出了数据库中所有的全文索引的列。例如：

```
SELECT object_name(ic.object_id) tblname, c.name
FROM sys.fulltext_index_columns ic
INNER JOIN sys.columns c ON
    ic.object_id = c.object_id AND
    ic.column_id = c.column_id
```

这个查询返回表名和索引列的名称：

```
tblname     name
Document    DocumentSummary
Document    Document
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

也可以使用FULLTEXTCATALOGPROPERTY系统函数来返回指定目录的信息。语法如下所示：

```
FULLTEXTCATALOGPROPERTY ('catalog_name' ,'property')
```

225 该函数有两个参数，目录的名称和要评估的属性的名称。一些比较有用的属性选项在表6-5中有描述。

表6-5　FULLTEXTCATALOGPROPERTY属性选项

属　性	描　述
AccentSensitivity	返回1表示区分重音，0为不区分重音
IndexSize	以MB返回全文目录的大小
MergeStatus	当进行重组时返回1，未进行重组时返回0
PopulateStatus	返回数字值表示目录当前的填充状态——例如，0为空闲，1为正在进行填充，2为暂停，7为正在生成索引，8为磁盘满

在这个示例中，返回全文目录的填充状态：

```
SELECT FULLTEXTCATALOGPROPERTY ('cat_Production_Document','PopulateStatus')
PopulationStatus
```

这个查询返回0，表示空闲：

```
PopulationStatus
0
```

解析

这个技巧使用了三个不同的目录视图和系统函数来返回当前数据库中全文目录和全文索引的信息。为了跟踪它们的存在以及跟踪活动和设置的当前状态，你会需要这些信息。

6.1.6　从全文索引中去掉普通字符串

SQL Server 2008引入了标识出那些对全文索引搜索无益的普通字符串的能力。这些无益的字符串被称为非索引字（SQL Server的早期版本中称为干扰词），并且包含在非索引字表中。非索引字表包含一个或多个非索引字，以及用来在全文索引中连接。SQL Server为所有支持的语言提供了包含普通非索引字的系统默认非索引字表。

使用CREATE FULLTEXT STOPLIST命令来创建你自定义的非索引字表。语法如下：

```
CREATE FULLTEXT STOPLIST stoplist_name
[ FROM { [ database_name. ] source_stoplist_name } | SYSTEM STOPLIST ]
[ AUTHORIZATION owner_name ];
```

这个命令的参数在表6-6中有描述。

表6-6　CREATE FULLTEXT STOPLIST参数

参　数	描　述
stoplist_name	提供新的用户定义的非索引字表的名称
{[database_name.]source_stoplist_name}	可以从引用数据库名称，也可以引用从既有非索引字表复制的源非索引字表名称
SYSTEM STOPLIST	复制系统默认的非索引字表
AUTHORIZATION owner_name	定义数据库主体非索引字表的拥有者

226

在这个示例中，将创建新的、不是从既有的非索引字表复制而来的非索引字表（注意全文非索引

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

字表语句必须以分号[；]来结束)：

```
CREATE FULLTEXT STOPLIST TSQLRecipes;
```

为了确认新非索引字表的细节，可以查询sys.full_text_stoplists系统目录视图：

```
SELECT stoplist_id,name,principal_id
FROM sys.fulltext_stoplists
```

这个查询返回：

```
stoplist_id   name          principal_id
5             TSQLRecipes   1
```

创建完非索引字表之后，现在可以使用ALTER FULLTEXT STOPLIST命令来填充它。这个命令的语法如下：

```
ALTER FULLTEXT STOPLIST stoplist_name
{   ADD 'stopword' LANGUAGE language_term
  | DROP
    {
              'stopword' LANGUAGE language_term
      | ALL LANGUAGE language_term
          | ALL };
```

这个命令的参数在表6-7中有描述。

表6-7 ALTER FULLTEXT STOPLIST参数

参 数	描 述
stoplist_name	指定新用户定义的非索引字表的名称
ADD 'stopword'	定义非索引字的字符串值。最长可以添加64个字符
LANGUAGE language_term	定义与非索引字关联的语言——可以是字符串（从sys.syslanguages中的别名）、整数（LCID）或十六进制表示（LCID的十六进制值）
DROP 'stopword' LANGUAGE language_term	指定删除指定的语言下指定的非索引字
DROP ALL LANGUAGE language_term	删除指定语言下的所有非索引字
DROP ALL	指定从非索引字表中删除所有非索引字

在这个示例中，假定我为包含引用了SQL Server文档参考书的表生成索引。在这个例子中，术语“SQL”和“Server”在搜索的上下文中不是非常有用（几乎所有条目都会包含它）。因此在这个示例中，我将增加两个新的非索引字到之前创建的非索引字表中：

```
ALTER FULLTEXT STOPLIST TSQLRecipes
ADD 'SQL' LANGUAGE 'English';

ALTER FULLTEXT STOPLIST TSQLRecipes
ADD 'Server' LANGUAGE 'English';
```

在增加两个新非索引字到非索引字表中之后，可以通过查询sys.fulltext_stopwords系统目录视图来验证这个列表：

```
SELECT stoplist_id,stopword,language
FROM sys.fulltext_stopwords
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
stoplist_id    stopword    language
5              SQL         English
5              Server      English
```

在下一个查询中，我将演示将新非索引字表绑定到全文索引上：

```
--示例表
CREATE TABLE dbo.SQLTopic
  (SQLTopic int IDENTITY PRIMARY KEY,
   SQLTopicHeaderNM varchar(255) NOT NULL,
   SQLTopicBody varchar(max) NOT NULL)
GO

-- 创建示例目录
CREATE FULLTEXT CATALOG ftcat_SQLDocumentation
AS DEFAULT
GO

-- 创建要绑定到我们的新非索引字表的全文索引
-- 使用sp_help 'dbo.sqltopic'来查看你实际的主键约束名
CREATE FULLTEXT INDEX ON dbo.SQLTopic(SQLTopicBody)
   KEY INDEX PK__SQLTopic__AD5554EC442B18F2
   WITH STOPLIST = TSQLRecipes
GO
```

可以使用sys.fulltext_indexs系统目录视图来确认绑定的非索引字表：

```
SELECT stoplist_id
FROM sys.fulltext_indexes
WHERE object_id = object_id('dbo.SQLTopic')
```

这个查询返回：

```
stoplist_id
5
```

可以通过使用sys.dm_fts_parser动态管理视图来测试我的新非索引字是否可以被全文引擎识别。这个DMV的语法如下：

```
sys.dm_fts_parser('query_string', lcid, stoplist_id, accent_sensitivity)
```

第一个参数query_string，是你可能会用在全文索引搜索中的查询字符串。lcid是地区标识符，stoplist_id是非索引字表的唯一ID（你可以从sys.fulltext_stoplists中检索）。accent_sensitivity参数的值可以是1或0，表示你的搜索是否区分重音。如下的查询使用前面创建的非索引字表测试搜索短语SQL Server 2008 Transact-SQL Recipes来演示这个DMV：

228

```
 SELECT display_term, special_term
FROM sys.dm_fts_parser
('"SQL Server 2008 Transact-SQL Recipes"', 1033, 5, 0)
```

结果返回了每个关键字的列表，以及它们被视为何种类型（干扰字/非索引字或精确匹配）：

```
display_term    special_term
sql             Noise Word
server          Noise Word
2008            Exact Match
nn2008          Exact Match
transact        Exact Match
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
sql             Noise Word
transactsql     Exact Match
recipes         Exact Match
```

正如你从结果中看到的，SQL和Server都被识别为干扰字（非索引字）。

在下一个查询中，演示从非索引字表中删除非索引字（就算非索引字表已经绑定到全文索引上也是可以的）：

```
ALTER FULLTEXT STOPLIST TSQLRecipes
DROP 'Server' LANGUAGE 'English';
```

使用DROP FULLTEXT STOPLIST命令来删除非索引字表。但是要想删除它，必须先使用它从全文索引中对非索引字表解除绑定。这个技巧的最后一个查询演示了从全文索引中移除非索引字表的设置，然后删除非索引字表：

```
ALTER FULLTEXT INDEX ON dbo.SQLTopic
SET STOPLIST SYSTEM
GO

DROP FULLTEXT STOPLIST TSQLRecipes;
```

解析

这个技巧演示了如何通过创建用户定义且包含非索引字列表的非索引字表从全文索引中去掉普通字符串。使用CREATE FULLTEXT STOPLIST命令来创建非索引字表。在创建索引字表之后，就可以使用ALTER FULLTEXT STOPLIST来增加非索引字字符串到非索引字表中，或从非索引字表中移除非索引字字符串。使用sys.fulltext_stoplists和sys.fulltext_stopwords系统目录视图来确认设置。然后创建新表和全文目录，再通过指定WITH STOPLIST = TSQLRecipes创建使用了新的非索引字的新全文索引。可以通过使用sys.dm_fts_parser来测试非索引字表中的非索引字是否被恰当地忽略。使用带有SET STOPLIST的ALTER FULLTEXT INDEX从全文索引中移除非索引字表，随后是DROP FULLTEXT STOPLIST命令。

6.1.7 删除全文索引

在这个技巧中，将示范如何用DROP FULLTEXT INDEX命令从全文目录中删除一个全文索引，语法如下所示：

```
DROP FULLTEXT CATALOG catalog_name
```

此命令只有一个参数，就是要删除的全文索引所在的表名称，例如：

```
DROP FULLTEXT CATALOG cat_Production_Document
```

解析

DROP FULLTEXT INDEX ON命令引用了全文索引的表名，由于在一个表中只允许存在一个全文索引，因此删除全文索引时不需要其他附加信息。

```
Warning: The fulltext catalog 'cat_Production_Document'
is being dropped and is currently set as default.
```

6.1.8 删除全文目录

在这个技巧中，将示范如何通过DROP FULLTEXT CATALOG命令从数据库中删除全文目录。语法如下所示：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DROP FULLTEXT INDEX ON table_name
```

这个命令只有一个参数，就是要删除的全文目录的名称，例如：

```
DROP FULLTEXT INDEX ON Production.Document
```

解析

DROP FULLTEXT CATALOG引用了全文目录的名称，而且从数据库中删除一个全文目录不需要更多的信息。假如这个全文目录被设置为数据库默认的全文目录，你会看到下面的警告：

```
警告: 正在删除全文目录'cat_Production_Document'，当前它被设置为默认值。
```

6.2　基本搜索

创建了全文目录和全文索引后，就可以使用更灵活的T-SQL谓词查询数据了。谓词用在WHERE或HAVING子句的表达式中或者FROM子句的联结条件中。谓词返回TRUE、FALSE或UNKNOWN值。

先从简单一点的命令开始，FREETEXT命令用来基于变形的、字面的、同义的匹配方式搜索非结构化的文本数据。它比使用LIKE的方式更智能，因为文本数据是按照意思而不是准确的单词来搜索。

CONTAINS谓词用于以精确或模糊的单词和短语匹配来搜索非结构化文本数据。这个命令还能考虑单词之间的接近程度，并且允许加权结果。

下面两个技巧会演示使用FREETEXT和CONTAINS谓词进行基本搜索。示例依赖于Production.Document表的DocumentSummary列的全文索引。在继续这个技巧之前先要创建这个索引：

```
USE AdventureWorks
GO
CREATE FULLTEXT CATALOG cat_Production_Document
CREATE FULLTEXT INDEX ON Production.Document
(DocumentSummary)
KEY INDEX PK_Document_DocumentNode
ON cat_Production_Document
WITH CHANGE_TRACKING AUTO,
     STOPLIST = SYSTEM
```

230

6.2.1　使用 FREETEXT 来搜索全文索引的列

FREETEXT命令用来基于变形的、字面的、同义的匹配方式搜索全文列。语法如下：

```
FREETEXT ( { column_name | (column_list) | * }
          , 'freetext_string' [ , LANGUAGE language_term ] )
```

表6-8描述了此谓词的参数。

表6-8　FREETEXT参数

参　数	描　述
column_name \| column_list \| *	希望搜索的带全文索引的一列或者多列的名字。*用来指定所有可以搜索的列
freetext_string	要搜索的文本
language_term	指示SQL Server使用某种语言来执行搜索、访问分类词典信息并移除干扰词

在这个示例中，使用FREETEXT来根据搜索条件的含义搜索数据。SQL Server查看各个单词，并且根据某种语言的分类词典搜索准确匹配、变形或扩展/同义词：

```
SELECT DocumentNode, DocumentSummary
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FROM Production.Document
WHERE FREETEXT (DocumentSummary, 'change pedal' )
```

这个查询返回：

```
DocumentNode    DocumentSummary
0x7BC0          Detailed instructions for replacing pedals with Adventure Works Cycles
                replacement pedals. Instructions are applicable to all Adventure Works
                Cycles bicycle models and replacement pedals. Use only Adventure Works
                Cycles parts when replacing worn or broken components.
```

解析

在这个技巧中，使用FREETEXT从DocumentSummary列搜索短语change pedal。尽管change和pedal两个单词在数据中都不存在，但是还是返回了一行，因为找到了一个pedal的复数形式（pedals）。

然而，和CONTAINS相比，FREETEXT是以比较模糊的方式来搜索全文索引。在后面几个技巧中我们会介绍CONTAINS。

231

6.2.2 使用 CONTAINS 来搜索单词

在这个技巧中，会演示使用CONTAINS命令来实现单词搜索。CONTAINS能以比FREETEXT谓词更灵活的全文项来搜索。基本语法如下：

```
CONTAINS
( { column_name | (column_list) | * } ,
'< contains_search_condition >'   [ , LANGUAGE language_term ] )
```

参数和FREETEXT一样，只是CONTAINS允许各种搜索条件（一些条件在6.3节中会介绍）。

这个示例演示了搜索DocumentSummary中带有单词replacing或pedals的行：

```
SELECT DocumentNode, DocumentSummary
FROM Production.Document
WHERE CONTAINS (DocumentSummary, '"replacing" OR "pedals"' )
```

这个查询返回：

```
DocumentNode    DocumentSummary
0x7BC0          Detailed instructions for replacing pedals with Adventure Works Cycles
                replacement pedals. Instructions are applicable to all Adventure Works
                Cycles bicycle models and replacement pedals. Use only Adventure Works
                Cycles parts when replacing worn or broken components.
0x7C20          Worn or damaged seats can be easily replaced following these simple
                instructions. Instructions are applicable to these Adventure Works Cycles
                models: Mountain 100 through Mountain 500. Use only Adventure Works Cycles
                parts when replacing worn or broken components.
```

解析

在这个技巧中，对DocumentSummary执行了一次搜索，找出任何包含replacing或pedals单词的摘要信息。和FREETEXT不同，这次是搜索字面单词而不是同义词或变形形式。

OR用于搜索包含这些单词中的任何一个的行，而AND能用于返回那些所有词都在DocumentSummary值中存在的行。

提示 对于一个单词的搜索项，双引号不是必需的，只需要一组单引号。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

6.3　高级搜索

至此，本章已经演示了一些相当简单易懂的单词搜索的示例。然而，使用CONTAINS我们还能对单词或短语执行更高级的搜索。包括：

- 使用通配符搜索来匹配某个文本前缀的单词或短语；
- 根据某个单词的变形形式搜索单词或短语；
- 根据单词之间的相邻搜索单词或短语。

232

接下来的3个技巧会演示使用CONTAINS谓词进行的3种更高级的搜索。

6.3.1　使用 CONTAINS 和通配符来搜索

在这个技巧中，会演示如何在CONTAINS搜索中使用通配符。在星号通配符之前指定前缀项：

```
SELECT DocumentNode, DocumentSummary
FROM Production.Document
WHERE CONTAINS (DocumentSummary, '"import*"' )
```

这个查询返回：

```
DocumentNode    DocumentSummary
0x5B40          It is important that you maintain your bicycle and keep it in good repair.
                Detailed repair and service guidelines are provided along with instructions
                for adjusting the tightness of the suspension fork.
```

解析

这个技巧使用星号作为一个或多个字符的通配符。这和使用LIKE差不多，只是可以从全文索引的固有性能中得益。任何匹配由import开始的单词的行都会返回。在本例中，返回了一行，其中的important单词有匹配。

如果使用了通配符，条件必须包括在双引号内，否则SQL Server会把星号当作文本值来搜索。例如，搜索不带双引号的‘import*’就等于把字面的星号值当作搜索条件的一部分来处理。

6.3.2　使用 CONTAINS 来搜索变形匹配

在这个技巧中，会演示如何基于变形形式来搜索匹配搜索项的行。搜索变形形式的基本语法如下：

```
FORMSOF ( { INFLECTIONAL | THESAURUS } , < simple_term > [,...n ] )
```

在这个示例中，我们搜索replace的变形形式：

```
SELECT DocumentNode, DocumentSummary
FROM Production.Document
WHERE CONTAINS(DocumentSummary, ' FORMSOF (INFLECTIONAL, replace) ')
```

这个查询返回：

```
DocumentNode    DocumentSummary
0x7B40          Reflectors are vital safety components of your bicycle. Always ensure your
                front and back reflectors are clean and in good repair. Detailed
                instructions and illustrations are included should you need to replace the
                front reflector or front reflector bracket of your Adventure Works Cycles
                bicycle.
0x7BC0          Detailed instructions for replacing pedals with Adventure Works Cycles
                replacement pedals. Instructions are applicable to all Adventure Works
                Cycles bicycle models and replacement pedals. Use only Adventure Works
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

233

0x7C20	Cycles parts when replacing worn or broken components. Worn or damaged seats can be easily replaced following these simple instructions. Instructions are applicable to these Adventure Works Cycles models: Mountain 100 through Mountain 500. Use only Adventure Works Cycles parts when replacing worn or broken components.

解析

这个技巧搜索**replace**的所有变形形式。尽管文本值不能总在列中找到，但是包含**replace*d***或者**replac*ing***的行还是会被返回。

THESAURUS是FORMSOF子句的另外一种选项，它能让我们根据同义项来搜索（在$SQL_Server_Install_Path\Microsoft SQL Server\<InstancePath>\MSSQL\FTDATA\目录中的XML文件中维护）。例如，法语的分类词典XML文件叫做tsFRA.xml。这些XML文件是可更新的，因此你可以根据自己应用程序的需求来定制它们。

6.3.3 使用 CONTAINS 根据词的相邻搜索结果

这个技巧演示如何使用CONTAINS来找到某个单词和另外一个互相接近的行。语法如下：

```
{ < simple_term > | < prefix_term > }
    { NEAR | ~ }
{ < simple_term > | < prefix_term > }
```

在这个示例中，单词**oil**和**grease**相邻的行被返回：

```
SELECT DocumentSummary
FROM Production.Document
WHERE CONTAINS(DocumentSummary, 'oil NEAR grease')
```

这个查询返回：

DocumentSummary
Guidelines and recommendations for lubricating the required components of your Adventure Works Cycles bicycle. Component lubrication is vital to ensuring a smooth and safe ride and should be part of your standard maintenance routine. Details instructions are provided for each bicycle component requiring regular lubrication including the frequency at which oil or grease should be applied.

解析

这个技巧寻找**grease**这个词和**oil**这个词相邻的所有文本。这个示例根据两个单词的相邻进行搜索，你还可以测试多个单词的相邻，例如：

```
SELECT DocumentSummary
FROM Production.Document
WHERE CONTAINS(DocumentSummary, 'oil NEAR grease AND frequency')
```

在这种情况下，所有三个词应彼此靠近。

234

6.4 排名搜索

前面的几个示例演示了在SELECT查询的WHERE子句中进行全文索引搜索。SQL Server同样支持在查询的FROM子句中使用排名函数。排名函数CONTAINSTABLE和FREETEXTTABLE不返回符合搜索条件的行，它们用来根据相关性返回指定的行。越是接近匹配，系统就给予越高的排名，接下来的两个技巧会演示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

6.4.1　根据含义返回排名搜索结果

在这个技巧中，会演示FREETEXTTABLE，它能根据搜索字符串按照排名的次序返回搜索结果。FREETEXT和FREETEXTTABLE的语法以及功能仍然很相似：

```
FREETEXTTABLE (table , { column_name | (column_list) | * }
          , 'freetext_string'
     [, LANGUAGE language_term ]
     [ ,top_n_by_rank ] )
```

FREETEXTTABLE和FREETEXT的区别就在于table和top_n_by_rank这两个参数。table参数是包含全文索引的一列或者多列的表名。如果指定top_n_by_rank参数，则它会有一个整数值，表示按照排名的最高几个匹配。

在这个示例中，根据与搜索项接近程度的高低从Production.Document表返回搜索bicycle seat的行：

```
SELECT   f.RANK, DocumentNode, DocumentSummary
FROM Production.Document d
INNER JOIN FREETEXTTABLE(Production.Document, DocumentSummary, 'bicycle seat') f
   ON d.DocumentNode = f.[KEY]
ORDER BY RANK DESC
```

这个查询返回：

RANK	DocumentNode	DocumentSummary
61	0x7C20	Worn or damaged seats can be easily replaced following these simple instructions. Instructions are applicable to these Adventure Works Cycles models: Mountain 100 through Mountain 500. Use only Adventure Works Cycles parts when replacing worn or broken c
37	0x6B40	Guidelines and recommendations for lubricating the required components of your Adventure Works Cycles bicycle. Component lubrication is vital to ensuring a smooth and safe ride and should be part of your standard maintenance routine. Details instructions a
37	0x7B40	Reflectors are vital safety components of your bicycle. Always ensure your front and back reflectors are clean and in good repair. Detailed instructions and illustrations are included should you need to replace the front reflector or front reflector bracke
21	0x7BC0	Detailed instructions for replacing pedals with Adventure Works Cycles replacement pedals. Instructions are applicable to all Adventure Works Cycles bicycle models and replacement pedals. Use only Adventure Works Cycles parts when replacing worn or broken
21	0x5B40	It is important that you maintain your bicycle and keep it in good repair. Detailed repair and service guidelines are provided along with instructions for adjusting the tightness of the suspension fork.

235

解析

FREETEXTTABLE和FREETEXT相似，都是根据含义而不是文本值搜索全文索引的列。但是FREETEXTTABLE和FREETEXT的不同之处在于，它可以像表一样在FROM子句中引用，并且允许你使用它的KEY来联结数据。KEY和RANK是FREETEXTTABLE在结果集中返回的两列。KEY是定义在全文索引中的唯一键/主键，而RANK是一个行在结果集中正确性的评估值。

在这个技巧中，FREETEXTTABLE结果集搜索DocumentSummary列的bicycle seat值，并根据它的KEY值联结到Production.Document表的DocumentNode列：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
INNER JOIN FREETEXTTABLE(Production.Document,
DocumentSummary,
'bicycle seat') f
   ON d.DocumentNode = f.[KEY]
```

结果根据匹配的强度，按照降序返回RANK：

```
ORDER BY RANK DESC
```

6.4.2 根据权值返回排名搜索结果

在这个技巧中，会演示使用CONTAINSTABLE命令根据加权模式匹配来返回搜索结果。CONTAINSTABLE和FREETEXTTABLE一样都表现为一个表并且能被FROM子句引用。CONTAINSTABLE也和CONTAINS有一样的搜索功能及变化形式。

CONTAINS和CONTAINSTABLE都能指定行匹配的“权值”，让一个项比另外一个更重要，因此也会影响结果排名。这通过命令中的ISABOUT来实现，它为搜索项分配一个权值。其基本语法如下：

```
ISABOUT { <search term> } [ WEIGHT ( weight_value ) ]
```

这个示例演示根据排名查询Production.Document，项**bicycle**比项**seat**有更高的权值：

```
SELECT f.RANK, d.DocumentNode, d.DocumentSummary
FROM Production.Document d
INNER JOIN CONTAINSTABLE(Production.Document, DocumentSummary,
'ISABOUT ( bicycle weight (.9), seat weight (.1))') f
   ON d.DocumentNode = f.[KEY]
ORDER BY RANK DESC
```

这个查询返回：

RANK	DocumentNode	DocumentSummary
23	0x6B40	Guidelines and recommendations for lubricating the required components of your Adventure Works Cycles bicycle. Component lubrication is vital to ensuring a smooth and safe ride and should be part of your standard maintenance routine. Details instructions a
23	0x7B40	Reflectors are vital safety components of your bicycle. Always ensure your front and back reflectors are clean and in good repair. Detailed instructions and illustrations are included should you need to replace the front reflector or front reflector bracke
11	0x7BC0	Detailed instructions for replacing pedals with Adventure Works Cycles replacement pedals. Instructions are applicable to all Adventure Works Cycles bicycle models and replacement pedals. Use only Adventure Works Cycles parts when replacing worn or broken
11	0x5B40	It is important that you maintain your bicycle and keep it in good repair. Detailed repair and service guidelines are provided along with instructions for adjusting the tightness of the suspension fork.

解析

CONTAINSTABLE是一个结果集，根据KEY和DocumentID来把它联结到Production.Document。在SELECT子句中返回RANK，并且在ORDER BY子句中根据它排序。CONTAINSTABLE可以实现和CONTAINS一样的搜索，包括通配符、相邻度、变形词以及分类词典搜索。

在这个示例中，实现了加权项搜索，也就是说单词都被指定了一个值来影响它们在结果排名中的权值。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在该技巧中，搜索了两个单词：bicycle和seat，bicycle的权值比较高：

```
'ISABOUT ( bicycle weight (.9), seat weight (.1))'
```

权值是0.0～1.0的数字，它影响每个行的匹配在CONTAINSTABLE中的排名。ISABOUT放在单引号中，列的定义放在圆括号中。每一项后都跟着单词weight和在圆括号中0.0～1.0的值。虽然权值不会影响从
237 查询返回的行，但它会影响排名值。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第7章 视 图

使用视图可以创建由SELECT语句定义的表数据的虚拟表现。定义的SELECT语句能联结一个或者多个表，能包含一个或者多个列。一旦创建，视图能在查询的FROM子句中被引用。

视图能用来为查询编写者简化数据访问，隐藏底层SELECT语句的复杂度。视图对管理安全和保护敏感数据同样有用。如果希望限制最终用户直接访问表，可以对视图授予执行权限，而不是对底层的表。还可以通过在视图定义中仅仅包含必要的列，暴露那些仅仅希望被最终用户看到的列。在某些情况下，视图甚至能允许直接的数据更新，这些在本章后面的7.1.7节有讲述。视图也对后台数据提供了标准的接口，除非底层的表结构有重大改动，否则你不需要对它们作任何修改。

除了普通视图，还可以创建索引视图。这些视图在数据库中保存了真实的索引数据（普通视图实际不会去保存物理数据）。还可以创建分布式分区视图，它可以让我们表示一个由分布在不同SQL Server实例中的水平分区表组成的逻辑表。表7-1列出了在SQL Server中使用的3种视图。

表7-1 SQL Server视图类型

视图类型	描 述
普通视图	这个视图由T-SQL查询定义。在数据库中不保存实际数据，只是视图定义
索引视图	这个视图一开始由T-SQL查询定义，然后在满足了一定条件后，就会创建聚集索引来物化和表数据相似的索引数据。一旦聚集索引创建后，就可以在需要时在索引视图上创建多个非聚集索引
分布式分区视图	出于性能或可扩展性（例如，扩展了每个SQL Server实例的表大小）的需求，这种视图使用UNION ALL来把多个跨两个以上SQL Server实例的较小的表组合成一个虚拟表

在本章中，我会介绍一些创建这3种类型的视图的技巧，同样也会介绍查看视图元数据的一些方法。

7.1 普通视图

视图是在把数据展现给最终用户之前预先筛选数据和列的好方法。视图能用于隐藏众多表联结和列的选取，也可以通过只对用户授权访问视图而不是实际的基础表来实现安全。

说了这么多视图的用途，我们仍然需要注意一些性能问题。如果考虑为你的数据库使用视图，考虑下面的最佳实践。

- 像SELECT查询一样对视图进行性能调优，因为普通视图从本质上说是一个“已存储的”查询。性能糟糕的视图会严重影响服务器性能。
- 不要嵌套视图超过一级。具体来说，不要定义一个调用另外一个视图的视图，等等。当你希望对性能低下的查询进行调优时，这么做会引起混乱，并且会降低每个嵌套级别的性能。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 如果可能的话，使用存储过程而不用视图。因为存储过程能重用执行计划，性能会有提升。存储过程也能减少网络流量，并可以实现更复杂的业务逻辑，而且比视图有更少的编码限制（更多信息请参阅第10章）。

创建视图后，它的定义就保存在数据库中，但是视图返回的真实数据不会从底层表分离出来。接下来的几个技巧会演示如何创建和管理视图。

7.1.1 创建基本视图

使用CREATE VIEW命令来创建视图。语法如下：

```
CREATE VIEW [ schema_name . ] view_name [ (column [ ,...n ] ) ]
[ WITH [ ENCRYPTION ] [ SCHEMABINDING ] [ VIEW_METADATA ] [ ,...n ] ]
AS select_statement
[ WITH CHECK OPTION ]
```

表7-2描述了这个命令的参数。在本章后面还会更详细地介绍其中的一些参数。

表7-2　CREATE VIEW参数

参　数	描　述
[schema_name .] view_name	这个参数指定新视图的架构和名字
(column [, ...n])	给视图使用的可选的列名列表。如果没有指定，会使用在SELECT查询中使用的名字（除非没有指定列的名称，而且这样会产生错误）
ENCRYPTION	加密系统表中的T-SQL视图定义，这样，如果没有保存原始CREATE VIEW命令的副本则不能看到这个定义
SCHEMABINDING	SCHEMABINDING将视图绑定到底层表的架构上，限制基础表中任何会影响视图定义的修改操作
VIEW_METADATA	如果指定了，则API访问有关视图信息的时候会看到视图的信息而不是基础表的元数据
select_statement	这个参数指定了用于返回视图的行或列的SELECT查询

240

SELECT语句允许一个视图定义最多1024个列。然而，不可以在视图定义中使用某些SELECT元素，包括INTO、OPTION、COMPUTE、COMPUTE BY或者对表变量或临时表的引用。除非使用了TOP关键字，否则你也不可以使用ORDER BY。

这个示例演示了如何创建一个从Production.TransactionHistory和Production.Product表中访问数据的视图：

```
USE AdventureWorks
GO

CREATE VIEW dbo.v_Product_TransactionHistory
AS

SELECT   p.Name ProductName,
      p.ProductNumber,
      c.Name ProductCategory,
      s.Name ProductSubCategory,
      m.Name ProductModel,
      t.TransactionID,
      t.ReferenceOrderID,
      t.ReferenceOrderLineID,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
      t.TransactionDate,
      t.TransactionType,
      t.Quantity,
      t.ActualCost
FROM Production.TransactionHistory t
INNER JOIN Production.Product p ON
   t.ProductID = p.ProductID
INNER JOIN Production.ProductModel m ON
   m.ProductModelID = p.ProductModelID
INNER JOIN Production.ProductSubcategory s ON
   s.ProductSubcategoryID = p.ProductSubcategoryID
INNER JOIN Production.ProductCategory c ON
   c.ProductCategoryID = s.ProductCategoryID
WHERE c.Name = 'Bikes'
GO
```

接着，查询新的视图，根据产品名和产品类型显示产品的交易记录：

```
SELECT ProductName, ProductModel, ReferenceOrderID, ActualCost
FROM dbo.v_Product_TransactionHistory
ORDER BY ProductName
```

这个查询返回了如下的部分结果：


```
ProductName              ProductModel ReferenceOrderID  ActualCost
Mountain-200 Black, 38   Mountain-200 53457            1652.3928
Mountain-200 Black, 38   Mountain-200 53463            1652.3928
...
Touring-3000 Yellow, 62  Touring-3000 71818            534.492
Touring-3000 Yellow, 62  Touring-3000 71822            534.492

(25262 行受影响)
```

解析

在这个技巧中，使用一个SELECT查询定义了一个视图，该查询在FROM子句中引用了多个表，并限制了产品分类为Bikes。这样的话，任何需要写查询访问这个数据的人都能从视图中受益了，因为用户不需要在每次写查询的时候指定很多表联结。

在这个视图的定义中我们也使用了列别名，使用ProductName来替代Name——使列名更明确，并减少了和其他叫Name的列发生混淆的可能。在视图的WHERE子句中限制从视图返回的数据同样限制了查询编写者能看到的数据——在这里，只允许查询编写者引用某种产品分类的产品。

7.1.2 查询视图定义

可以通过查询sys.sql_modules系统目录视图来查看视图的T-SQL定义。

这个示例显示了如何查询某个视图的SQL定义：

```
SELECT definition FROM sys.sql_modules
WHERE object_id = OBJECT_ID('v_Product_TransactionHistory')
```

这个查询返回：

```
definition
CREATE VIEW dbo.v_Product_TransactionHistory AS
SELECT p.Name ProductName,
        p.ProductNumber,
        c.Name ProductCategory,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        s.Name ProductSubCategory,
        m.Name
        ProductModel,
        t.TransactionID,
        t.ReferenceOrderID,
        t.ReferenceOrderLineID,
        t.TransactionDate,
        t.TransactionType,
        t.Quantity,
        t.ActualCost
FROM Production.TransactionHistory t
INNER JOIN Production.Product p ON
   t.ProductID = p.ProductID
INNER JOIN Production.ProductModel m ON
   m.ProductModelID = p.ProductModelID
INNER JOIN Production.ProductSubcategory s ON
   s.ProductSubcategoryID = p.ProductSubcategoryID
INNER JOIN Production.ProductCategory c ON
   c.ProductCategoryID = s.ProductCategoryID
WHERE c.Name = 'Bikes'
```

242

解析

就像刚才看到的那样，sys.sql_modules系统目录视图和OBJECT_DEFINITION函数允许查看视图的SQL文本。对于加密过的视图（7.2节会介绍加密），definition列会返回NULL值。在后面的章节你还会看到使用这个系统目录视图来查看诸如触发器、函数和存储过程等其他过程代码对象类型。

7.1.3 显示视图及其结构

在这个技巧中，我会使用3个不同的查询来返回当前数据库中有关视图的信息。

第一个查询显示了当前数据库中所有的视图：

```
SELECT   s.name SchemaName,
      v.name ViewName
FROM sys.views v
INNER JOIN sys.schemas s ON
   v.schema_id = s.schema_id
ORDER BY s.name,
       v.name
```

这个查询返回了如下的（部分）结果：

```
SchemaName          ViewName
dbo                 v_Product_TransactionHistory
HumanResources      vEmployee
HumanResources      vEmployeeDepartment
...
Sales               vStoreWithAddresses
Sales               vStoreWithContacts
Sales               vStoreWithDemographics

(21行受影响)
```

第二个查询显示了当前数据库中每个视图公开的列：

```
SELECT  v.name ViewName,
      c.name ColumnName
FROM sys.columns c
INNER JOIN sys.views v ON
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
   c.object_id = v.object_id
ORDER BY v.name,
      c.name
```

243

该查询返回如下的（部分）结果：

```
ViewName                        ColumnName
v_Product_TransactionHistory    ActualCost
v_Product_TransactionHistory    ProductCategory
v_Product_TransactionHistory    ProductModel
v_Product_TransactionHistory    ProductName
...
vVendorWithContacts             PhoneNumberType
vVendorWithContacts             Suffix
vVendorWithContacts             Title

(270行受影响)
```

解析

技巧中的第一个查询引用了对象分类视图sys.views和sys.schemas来返回数据库中的所有视图：

```
FROM sys.views v
INNER JOIN sys.schemas s ON
   v.schema_id = s.schema_id
```

第二个查询通过查询对象分类视图sys.columns和sys.views来统计每个视图返回的所有列：

```
FROM sys.columns c
INNER JOIN sys.views v ON
   c.object_id = v.object_id
```

提示 视图可以在视图定义中引用其他视图或表。被引用的对象是依赖对象（视图依赖它们返回数据）。使用sys.sql_expression_dependencies目录视图来查询视图的对象依赖性。这个目录视图是SQL Server 2008新带来的，我将在第24章讲解它。

7.1.4 刷新视图定义

当视图引用的表对象修改后，视图的元数据可能就过时了。例如，如果更改在视图定义中引用的列的宽度，那么在刷新元数据之前新的宽度可能不会生效。在这个技巧中，会演示在视图定义中引用的依赖对象修改后，如何刷新视图的元数据：

```
EXEC sp_refreshview 'dbo.v_Product_TransactionHistory'
```

也可以使用系统存储过程sp_refreshsqlmodule，它不仅仅可以使用在视图上，也可以使用到存储过程、触发器和用户定义函数上：

```
EXEC sys.sp_refreshsqlmodule @name = 'dbo.v_Product_TransactionHistory'
```

244

解析

如果视图的SELECT查询定义引用的基础对象修改了，可以使用sp_refreshview或sys.refreshsqlmodule存储过程来刷新视图的元数据。这个系统存储过程只接受一个参数，那就是视图的名称。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

7.1.5　修改视图

ALTER VIEW命令用来修改既有视图的定义。语法如下：

```
ALTER VIEW [ schema_name . ] view_name [ ( column [ ,...n ] ) ]
[ WITH [ ENCRYPTION ] [ SCHEMABINDING ] [ VIEW_METADATA ] [ ,...n ] ]
AS select_statement
[ WITH CHECK OPTION ]
```

ALTER VIEW和CREATE VIEW使用相同的参数。这个示例演示了修改既有视图：

```
-- 增加WHERE子句并移除ReferenceOrderID和ReferenceOrderLineID列

ALTER VIEW dbo.v_Product_TransactionHistory
AS

SELECT   p.Name,
      p.ProductNumber,
      t.TransactionID,
      t.TransactionDate,
      t.TransactionType,
      t.Quantity,
      t.ActualCost
FROM Production.TransactionHistory t
INNER JOIN Production.Product p ON
   t.ProductID = p.ProductID
WHERE Quantity > 10

GO
```

解析

这个技巧用来从原始视图中移除两列并增加一个WHERE子句——都是通过在ALTER VIEW命令中的AS关键字之后重新定义SELECT语句来进行的。注意，如果修改一个索引视图（indexed view）（本章后面介绍），所有索引都会被删除并需要手动重建。

7.1.6　删除视图

可以通过使用DROP VIEW命令来删除一个视图。语法如下：

```
DROP VIEW [ schema_name . ] view_name [ ,... n]
```

这个命令只接受一个参数，即需要从数据库中删除的一个或多个视图名。

这个示例演示了删除一个视图：

245

```
DROP VIEW dbo.v_Product_TransactionHistory
```

解析

删除视图会从系统目录中移除它的定义，如果它是索引视图，相关的所有索引也会被移除。

7.1.7　使用视图修改数据

在本章开始的时候提到过，可以像普通表那样对视图进行插入、更新和删除操作。要实现这些操作，任何INSERT/UPDATE/DELETE操作只能引用一个表中的列。而且，INSERT/UPDATE/DELETE中引用的列不能进行衍生——例如，它们不能基于聚合函数被计算或受GROUP BY、DISTINCT或HAVING子句的影响。

作为真实情况的最佳实践，只有当基础数据表必须对查询编写者不可见的时候，才应该使用视图更新。例如，如果你正在编写允许用户直接更新数据的套装软件应用程序，那么就能通过提供视图来

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

过滤查看的基础列或提供比基础表中使用的更友好的列名。

在本例中，我们创建一个从Production.Location表查询的视图。在查询定义中还使用了计算列：

```
CREATE VIEW Production.vw_Location
AS
SELECT LocationID,
   Name LocationName,
   CostRate,
   Availability,
  CostRate/Availability CostToAvailabilityRatio
FROM Production.Location
GO
```

尝试使用下面的插入：

```
INSERT Production.vw_Location
(LocationName, CostRate, Availability, CostToAvailabilityRatio)
VALUES ('Finishing Cabinet', 1.22, 75.00, 0.01626 )
```

这个插入返回如下的错误：

```
消息4406，级别16，状态1，第1行
对视图或函数 'Production.vw_Location' 的更新或插入失败，因其包含派生域或常量域。
```

尝试下面的插入，这次我们仅引用在基础表中存在的列：

```
INSERT Production.vw_Location
(LocationName, CostRate, Availability)
VALUES ('Finishing Cabinet', 1.22, 75.00)
```

结果表明插入成功了：

```
(1行受影响)
```

解析

在这个技巧中演示了对视图实现插入操作。只要数据修改和视图满足要求，就可以对视图进行数据修改操作。如果视图不能满足这些要求，可以使用INSTEAD OF触发器来进行更新（在触发器上创建视图的示例会在第12章演示）。

7.2 视图加密

CREATE VIEW和ALTER VIEW命令中的ENCRYPTION OPTION允许我们加密视图的T-SQL。一旦加密，就不能再通过sys.sql_modules系统目录视图查看视图定义。在后端使用SQL Server的软件厂商经常加密它们的视图或存储过程，目的是防止客户端或竞争对手篡改或进行逆向工程。如果使用加密，请确保保存了原始的、未加密的定义。

加密视图

这个示例演示了加密新视图的T-SQL定义：

```
CREATE VIEW dbo.v_Product_TopTenListPrice
WITH ENCRYPTION
AS

SELECT   TOP 10
      p.Name,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    p.ProductNumber,
    p.ListPrice
FROM Production.Product p
ORDER BY p.ListPrice DESC
GO
```

然后，通过查询sys.sql_modules系统目录视图来查看新视图的T-SQL定义：

```
SELECT definition
FROM   sys.sql_modules
WHERE object_id = OBJECT_ID('v_Product_TopTenListPrice')
```

这个查询返回：

```
definition
NULL
```

解析

在这个技巧中，我们使用WITH ENCRYPTION选项创建了一个新的视图。如果使用这个选项，请确保在安全的地方保存了源代码，或使用了诸如Visual SourceSafe的版本控制程序。通常，如果必须加密视图定义，应该在部署前进行。

247

7.3　索引视图

视图并不比定义它的底层SELECT查询更有效。然而，一个可以提升经常被访问的视图性能的方法就是为之添加索引。要这么做，必须先在视图上创建一个唯一的聚集索引。一旦视图上的索引被创建，用于物化视图的数据就像表的聚集索引那样保存。在视图上创建了唯一的聚集索引之后，你还可以创建另外的非聚集索引。基础表不会受到这些视图索引创建的影响，因为它们是独立的基础对象。

索引视图能在任何版本的SQL Server中创建，但是只有SQL Server企业版的查询优化器能自动在查询执行计划中考虑使用索引视图。在SQL Server企业版中，如果查询优化器觉得索引视图有用，即使SQL语句显示引用了视图的底层基础表而不是视图本身，查询优化器也会自动使用它。对于企业版之外的版本，可以通过使用NOEXPAND表提示手动强制查询优化器使用索引视图（7.3.2节会讲解该内容）。

索引视图对于跨多行聚合数据的视图定义来说特别理想，因为聚合值能保持是最新的和物化的，可以直接查询而不用重新计算。索引视图对频繁引用更新的基础表的查询来说也很理想，但是在变化非常快的表上创建它们可能会由于要不断更新索引而导致性能下降。更新非常频繁的基础表会引起视图的索引频繁更新，也就是说更新速度要以查询性能为代价。

7.3.1　创建索引视图

在这个技巧中，会演示如何创建索引视图。首先，会新建一个视图，然后再为它创建索引（聚集和非聚集）。要创建索引视图，需要使用新的WITH SCHEMABINDING选项，它绑定视图到基础表的架构上。这样会阻止在基础表上进行的、对视图定义有影响的任何修改操作。WITH SCHEMABINDING选项也为视图的查询定义增加了额外的需求。架构绑定视图中引用的对象必须包含两部分的schema.object命名规则，并且所有引用的对象必须处于同一个数据库中。

注解　还有一些其他的需求，它们可以决定是否要在视图上创建索引。完整的列表不会在本章重复了，所以要确保检查了SQL Server联机丛书中的完整需求。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个技巧由使用SCHEMABINDING选项创建新的视图开始：

```
CREATE VIEW dbo.v_Product_Sales_By_LineTotal
WITH SCHEMABINDING
AS

SELECT p.ProductID, p.Name ProductName,
       SUM(LineTotal) LineTotalByProduct,
       COUNT_BIG(*) LineItems
FROM Sales.SalesOrderDetail s
INNER JOIN Production.Product p ON
   s.ProductID = p.ProductID
GROUP BY p.ProductID, p.Name
GO
```

248

在创建索引之前，我们会演示查询普通视图，使用SET STATISTICS IO命令来返回I/O成本的统计信息（更多有关这个命令的信息请参阅第28章）：

```
SET STATISTICS IO ON
GO

SELECT TOP 5 ProductName, LineTotalByProduct
FROM v_Product_Sales_By_LineTotal
ORDER BY LineTotalByProduct DESC
```

这个查询返回如下的结果：

```
ProductName              LineTotalByProduct
Mountain-200 Black, 38   4400592.800400
Mountain-200 Black, 42   4009494.761841
Mountain-200 Silver, 38  3693678.025272
Mountain-200 Silver, 42  3438478.860423
Mountain-200 Silver, 46  3434256.941928
```

7

它也返回了I/O信息，汇总了视图中使用的底层基础表的各种扫描行为（如果你跟着技巧一起做练习，记住你实际的统计信息可能会不同）：

```
表'Product'。扫描计数0，逻辑读取10次，物理读取0次，预读0次，lob逻辑读取0次，lob物理读取0次，lob预读0次。

表'Worktable'。扫描计数0，逻辑读取0次，物理读取0次，预读0次，lob逻辑读取0次，lob物理读取0次，lob预读0次。

表'SalesOrderDetail'。扫描计数1，逻辑读取1241次，物理读取7次，预读1251次，lob逻辑读取0次，lob物理读取0次，
lob预读0次。
```

然后，在普通视图上根据ProductID视图列的唯一值创建一个聚集索引：

```
CREATE UNIQUE CLUSTERED INDEX UCI_v_Product_Sales_By_LineTotal
  ON dbo.v_Product_Sales_By_LineTotal (ProductID)
GO
```

一旦创建了聚集索引，然后就可以开始根据需要创建非聚集索引：

```
CREATE NONCLUSTERED INDEX NI_v_Product_Sales_By_LineTotal
  ON dbo.v_Product_Sales_By_LineTotal (ProductName)
GO
```

接着，把前面在普通视图上执行的查询在索引视图上再执行一次：

```
SELECT TOP 5 ProductName, LineTotalByProduct
FROM v_Product_Sales_By_LineTotal
ORDER BY LineTotalByProduct DESC
```

249

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回了和先前一样的结果，只是这次的I/O行为不同。以前是访问了两个基础表和一个工作表（用来临时处理结果的tempdb），这次只访问了一个对象来获取结果：

```
表'v_Product_Sales_By_LineTotal'。扫描计数1，逻辑读取5次，物理读取0次，预读0次，
lob逻辑读取0次，lob物理读取0次，lob预读0次。
```

解析

索引视图允许我们把视图的结果物化成一个物理对象，和普通表以及相关的索引相似。它允许SQL Server查询优化器从单独的物理区域获取数据，而不用每次调用的时候处理视图定义查询。

在本例中，我们使用SCHEMABINDING选项创建了一个视图：

```
CREATE VIEW dbo.v_Product_Sales_By_LineTotal
WITH SCHEMABINDING
AS
```

视图余下的部分是普通的SELECT查询，它根据产品来聚合销售总数：

```
SELECT p.ProductID, p.Name ProductName,
      SUM(LineTotal) LineTotalByProduct,
      COUNT_BIG(*) LineItems
FROM Sales.SalesOrderDetail s
INNER JOIN Production.Product p ON
   s.ProductID = p.ProductID
GROUP BY p.ProductID, p.Name

GO
```

注意到，查询引用了COUNT_BIG聚合函数。需要COUNT_BIG是因为SQL Server要维护索引视图中每一个分组的行数。一旦视图使用SCHEMABINDING成功创建，就会在视图上创建唯一的聚集索引：

```
CREATE UNIQUE CLUSTERED INDEX UCI_v_Product_Sales_By_LineTotal
  ON dbo.v_Product_Sales_By_LineTotal (ProductID)
GO
```

要索引一个视图，就必须先在它上面创建唯一的聚集索引。一旦建立了索引，视图数据就像为表创建的聚集索引那样被保存。创建了聚集索引之后，你还可以创建另外的非聚集索引，就像为普通表创建那样。本例中，我们在索引视图的ProductName列上创建了非聚集索引：

```
CREATE NONCLUSTERED INDEX NI_v_Product_Sales_By_LineTotal
  ON dbo.v_Product_Sales_By_LineTotal (ProductName)
GO
```

一旦视图被索引了，只要视图或者基础表被查询引用，SQL Server企业版就能使用视图索引。我
250 们使用SET STATISTICS IO命令来演示SQL Server在视图索引前后如何进行数据页获取。

索引视图对于相对静态的数据能提供更好的性能。从另外一面来说，经常更新的基础表不适合被索引视图引用，因为更新同样会引起视图索引的频繁更新，从而潜在地减少查询性能上的获益。这是数据修改速度和查询速度的折中。

尽管索引视图能使用任何版本的SQL Server创建，但如果使用企业版的话，它们会在查询中自动被考虑使用。要确保其他版本的SQL Server使用，你需要使用视图提示NOEXPAND，下一个技巧会介绍。

7.3.2　强制优化器为索引视图使用索引

一旦创建了索引视图，如果使用的是SQL Server企业版，查询优化器会自动决定是否需要在查询

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

中使用索引视图。然而，对于其他的版本，要使SQL Server使用某个索引视图，必须使用NOEXPAND关键字。

通过在FROM子句后增加WITH(NOEXPAND)视图提示，指示SQL Server只可以使用视图索引。

视图提示语法如下：

```
{ NOEXPAND [ , INDEX ( index_val [ ,...n ] ) ] }
```

这个技巧演示如何强制查询使用索引视图的索引：

```
SELECT ProductID
FROM dbo.v_Product_Sales_By_LineTotal
WITH (NOEXPAND)
WHERE ProductName = 'Short-Sleeve Classic Jersey, L'
```

NOEXPAND同样允许为查询指定一个或多个索引，使用INDEX选项。例如：

```
SELECT ProductID
FROM dbo.v_Product_Sales_By_LineTotal
WITH (NOEXPAND, INDEX(NI_v_Product_Sales_By_LineTotal))
WHERE ProductName = 'Short-Sleeve Classic Jersey, L'
```

解析

对于那些使用非企业版的SQL Server的人来说，仍然可以通过使用NOEXPAND关键字来利用索引视图。缺点是必须在利用索引视图的时候显式使用提示。另外一个缺点是如果提示没有被使用，你的提示使用可能会废除SQL Server查询优化器已经作出的选择。

7.4 分区视图

分布式分区视图允许你为分布在不同SQL Server实例的两个或多个水平分区表创建单个逻辑表现（视图）。

要创建一个分布式分区视图，需要根据CHECK约束中定义的一组值把大表分割成更小的一些表。CHECK约束确保每一个更小的表保存着不能在其他表中保存的唯一数据。然后使用UNION ALL创建分布式分区视图，把所有小表联结成单独的结果集。

对分布式分区视图执行查询能得到性能增益。例如，如果视图根据日期范围分区，并用查询来返回仅保存在一个分区表中的行，那么SQL Server会智能地只搜索一个分区而不是分布式分区视图中的所有表。

创建分布式分区视图

在这个技巧中，会演示如何创建跨两个SQL Server实例的分布式分区视图。它基于下面的业务应用。有两个兄弟公司——MegaCorp和MiniCorp。每一个都有它们自己的SQL Server实例来保存网站数据，并且每一个都希望有一个表来跟踪网站点击。点击量是非常大的——需要比单个SQL Server实例能处理的更多的空间。这就需要创建一个统一的视图以在单个视图中引用各自的表。公司希望能查询任意一个服务器，并且每一个返回相同的数据或自己公司的数据。

由于分布式分区技巧会访问多个SQL Server实例，因此我们为两个参与的SQL Server实例都增加了链接服务器（更多关于链接服务器的信息请参阅第27章）。

这个技巧从为第一个SQL Server实例增加链接服务器开始：

```
USE master
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
EXEC sp_addlinkedserver
   'JOEPROD',
   N'SQL Server'
GO

-- 跳过检查远程表架构
EXEC sp_serveroption 'JOEPROD', 'lazy schema validation', 'true'
GO
```

在第二个SQL Server实例上，创建一个到第一个SQL Server实例的链接服务器：

```
USE master
GO
EXEC sp_addlinkedserver
   'JOEPROD\SQL2008',
   N'SQL Server'
GO

-- 跳过检查远程表架构
EXEC sp_serveroption 'JOEPROD\SQL2008', 'lazy schema validation', 'true'
GO
```

回到第一个SQL Server实例，创建下面的表来保存MegaCorp网站点击的行：

```
IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'TSQLRecipeTest')
BEGIN
     CREATE DATABASE TSQLRecipeTest
END
GO

Use TSQLRecipeTest
GO

CREATE TABLE dbo.WebHits_MegaCorp
   (WebHitID uniqueidentifier NOT NULL,
    WebSite varchar(20) NOT NULL ,
    HitDT datetime NOT NULL,
    CHECK (WebSite = 'MegaCorp'),
    CONSTRAINT PK_WebHits PRIMARY KEY (WebHitID, WebSite))
```

252

在第二个SQL Server实例中创建下面的表来保存MiniCorp网站点击的行：

```
IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'TSQLRecipeTest')
BEGIN
     CREATE DATABASE TSQLRecipeTest
END
GO

USE TSQLRecipeTest
GO

CREATE TABLE dbo.WebHits_MiniCorp
   (WebHitID uniqueidentifier NOT NULL ,
    WebSite varchar(20) NOT NULL ,
    HitDT datetime NOT NULL,
    CHECK (WebSite = 'MiniCorp') ,
    CONSTRAINT PK_WebHits PRIMARY KEY (WebHitID, WebSite))
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

回到第一个SQL Server实例，创建引用本地的WebHits_MegaCorp表和远程的WebHits.MiniCorp表的分布式分区视图：

```
CREATE VIEW dbo.v_WebHits AS
  SELECT WebHitID,
       WebSite,
       HitDT
  FROM TSQLRecipeTest.dbo.WebHits_MegaCorp
UNION ALL
  SELECT WebHitID,
       WebSite,
       HitDT
  FROM JOEPROD.TSQLRecipeTest.dbo.WebHits_MiniCorp
GO
```

在第二个SQL Server实例上，创建下面的分布式分区视图——这次引用本地的WebHits_MiniCorp表和远程的WebHits_MegaCorp表：

```
CREATE VIEW dbo.v_WebHits AS
  SELECT WebHitID,
       WebSite,
       HitDT
  FROM TSQLRecipeTest.dbo.WebHits_MiniCorp
UNION ALL
  SELECT WebHitID,
       WebSite,
       HitDT
  FROM [JOEPROD\SQL2008].TSQLRecipeTest.dbo.WebHits_MegaCorp
GO
```


在第二个SQL Server实例上执行下面一组查询来插入新行：

```
-- 要让这些插入操作可行，必须把XACT_ABORT设置成ON，并开启分布式事务协调器

SET XACT_ABORT ON

INSERT dbo.v_WebHits
(WebHitID, WebSite, HitDT)
VALUES(NEWID(), 'MegaCorp', GETDATE())

INSERT dbo.v_WebHits
(WebHitID, WebSite, HitDT)
VALUES(NEWID(), 'MiniCorp', GETDATE())
```

这个查询返回：

```
(1行受影响)
(1行受影响)
```

查询分布式分区视图来返回插入的两个新行（从两个基础表）：

```
SET XACT_ABORT ON

SELECT WebHitID, WebSite, HitDT
FROM dbo.v_WebHits
```

这个查询返回：

```
WebHitID                             WebSite     HitDT
E5994678-6066-45F4-8AE4-9F10CE412D1A MegaCorp    2008-08-06 16:56:29.253
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
E1444A3F-7A2E-4A54-A156-C04FE742B453    MiniCorp    2008-08-06 16:56:29.353
```

查询MiniCorp表直接返回期望的一个MiniCorp行：

```
SELECT WebHitID, WebSite, HitDT
FROM JOEPROD.AdventureWorks.dbo.WebHits_MiniCorp
```

这个查询返回：

```
WebHitID                                WebSite    HitDT
E1444A3F-7A2E-4A54-A156-C04FE742B453    MiniCorp   2005-08-06 16:56:29.353
```

查询MegaCorp表同样返回期望的一行：

```
SELECT WebHitID, WebSite, HitDT
FROM [JOEPROD\SQL2008].AdventureWorks.dbo.WebHits_MegaCorp
```

这个查询返回：

```
WebHitID                                WebSite    HitDT
E5994678-6066-45F4-8AE4-9F10CE412D1A    MegaCorp   2005-08-06 16:56:29.253
```

254

解析

分布式分区视图允许我们跨多个SQL Server实例划分数据。对于超大型数据库和拥有大量事务和读操作的SQL Server实例来说，使用这样的设计能从中获益。

在这个技巧中做了很多，我会一点点介绍每一步。首先，在每个SQL Server实例上创建了链接服务器引用，这样每一个实例都能使用分布式查询来和另外一个通信（同样参阅第27章）。而且，由于性能原因开启了服务器选项Lazy schema validation。这个设置确保在查询执行之前不查询架构。

接着，在SQL Server实例1（JOEPROD\SQL2008）上创建dbo.WebHits_MegaCorp表，并在SQL Server实例2（JOEPROD）上创建dbo.WebHits_MiniCorp。每一个表创建时都带CHECK约束，用来限制可能会加入的值。所以这个分布式分区视图允许更新，CHECK约束必须定义在同一列上，并且不能允许在成员表中出现重复值。除此之外，只有<、>、=、>=、<=、AND、OR和BETWEEN运算符能用于CHECK约束。

要允许视图更新，你还需要记住一些其他需求：分区列（在这里是WebSite）不能允许空值，计算列或者标识、默认或时间戳列。分区键WebSite也需要是主键的一部分。由于WebSite本身不唯一，它和uniqueidentifier数据类型的WebHitID被作为组合键加入。两个分区表需要在同样数量的列上有主键：

```
CONSTRAINT PK_WebHits PRIMARY KEY (WebHitID, WebSite))
```

接下来，我们在每一个SQL Server实例上创建分布式分区视图。在拥有dbo.WebHits_MegaCorp表的实例上，视图使用3部分的格式database.schema.tablename引用了表（因为这是本地表）：

```
SELECT WebHitID,
       WebSite,
       HitDT
  FROM AdventureWorks.dbo.WebHits_MegaCorp
```

然后使用UNION ALL来联结表（如果希望对分布式分区视图进行数据修改的又一个需求）：

```
UNION ALL
```

定义在SELECT列表中的列不能在一个列表被引用多次，并且对于每一个UNION的SELECT来说应有相同的次序。跨每个SELECT的列必须有同样的数据类型和字符集，正如本技巧中所做的一样。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

对于远程的dbo.WebHits_MiniCorp表的FROM子句，使用4部分构成的名字linkedservername.database.schema.tablename（因为这是远程表）：

```
    SELECT WebHitID,
        WebSite,
        HitDT
    FROM JOEPROD.AdventureWorks.dbo.WebHits_MiniCorp
GO
```

在本技巧的最后一段代码中，把SET XACT_ABORT设置成ON来允许行插入到分布式分区视图中。如果出现了运行时错误，这个选项终止并回滚事务：

```
SET XACT_ABORT ON
```

255

在脚本注释中提到过，还需要开启分布式事务协调器（Distributed Transaction Coordinator），以在跨SQL Server实例插入行的时候调用分布式事务。对新的分布式分区视图进行了两次插入，第一次是对MegaCorp的点击，第二次是对MiniCorp的点击：

```
INSERT dbo.v_WebHits
(WebHitID, WebSite, HitDT)
VALUES(NEWID(), 'MegaCorp', GETDATE())

INSERT dbo.v_WebHits
(WebHitID, WebSite, HitDT)
VALUES(NEWID(), 'MiniCorp', GETDATE())
```

7

查询新的分布式分区视图，返回了两行：

```
SELECT WebHitID, WebSite, HitDT
FROM dbo.v_WebHits
```

查询基础的水平分区表，发现一行自动发送到了dbo.WebHits_MegaCorp表，而另外一行到了dbo.WebHits_MiniCorp表。

根据被查询的视图（例如实例1或实例2），SQL Server能确定只查询本地分区表是否能满足某个查询请求，远程表是否也需要查询。最终，SQL Server会最大限度地减少需要在SQL Server实例间传输的数据总量。

256

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第8章 SQL Server函数



在本章中，会演示如何在T-SQL代码中使用SQL Server内建函数。不要把SQL Server内建函数和第11章介绍的用户定义函数混淆，SQL Server内建函数能让我们实现聚合、数学运算、字符串操作、行排序等。SQL Server 2008也增加了新的函数，包括GROUPING、SYSDATETIME、SYSDATETIMEOFFSET、SYSUTCDATETIME、SWICHOFFSET和TODATETIMEOFFSET——我会演示所有这些新增函数。我也会介绍SQL Server 2008新带来的以字符串十六进制字面量格式返回二进制数据的增强功能。

8.1 聚合函数

聚合函数用于对一个或多个值进行计算，最后得出一个值。经常使用的聚合函数的一个例子是SUM，它用来返回一组数字值的总值。表8-1列出了SQL Server提供的一些经常使用的聚合函数。

表8-1 聚合函数

函数名	描述
AVG	AVG聚合函数计算一个组中非NULL值的平均值
CHECKSUM_AGG	CHECKSUM_AGG函数返回基于行组的校验和值，可以潜在地跟踪表的变化。例如，添加新行或改变要聚合的列值，通常会产生新的校验和整数值。提到“通常”的原因是即使值发生改变，校验和值也存在不变的可能
COUNT	COUNT聚合函数返回一个显示组中行数的整数值
COUNT_BIG	和COUNT功能一样，只是COUNT_BIG返回bigint数据类型值
GROUPING	GROUPING函数根据NULL值是否源于CUBE、ROLLUP或GROUPING SETS操作返回1（True）或0（False）。如果为False，表示列的NULL值从自然数据中返回参见1.11.4节
MAX	MAX聚合函数返回一组非NULL值中的最大值
MIN	MIN聚合函数返回一组非NULL值中的最小值
SUM	SUM聚合函数返回一个表达式中所有非NULL值的总和
STDEV	STDEV函数根据数据母体的样本返回由表达式提供的所有值的标准差
STDEVP	STDEVP函数也返回由表达式提供的所有值的标准差，只是它计算整个数据母体
VAR	VAR函数根据数据母体的样本返回由表达式提供的值的方差
VARP	VARP函数也返回表达式整个数据母体提供的值的方差

接下来的几个技巧会演示这些聚合函数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

8.1.1 返回平均值

AVG聚合函数计算一个组中非NULL值的平均值。例如：

```
-- 产品平均等级
SELECT ProductID,
      AVG(Rating) AvgRating
FROM Production.ProductReview
GROUP BY ProductID
```

这个查询返回：

```
ProductID       AvgRating
709             5
798             5
937             3
```

第二个示例演示对StrandardCost列的DISTINCT值取平均——也就是说只有唯一的StandardCost值才会被平均：

```
-- 对DISTINCT StandardCost求平均
SELECT AVG(DISTINCT StandardCost) AvgDistinctStandardCost
FROM Production.ProductCostHistory
```

这个查询返回：

```
AvgDistinctStandardCost
287.7111
```

解析

在这个技巧中，第一个示例返回由ProductID分组的平均产品等级。第二个示例对DISTINCT StandardCost进行平均——也就是说只有唯一的StandardCost才会被平均。没有DISTINCT关键字，AVG聚合函数默认的行为会对包括重复值在内的所有值取平均。

258

8.1.2 返回行总数

COUNT聚合函数返回一个显示组中行的总数的整数。例如，如下的查询通过货架对行分组，并返回每个货架的产品总数：

```
SELECT Shelf,
      COUNT(ProductID) ProductCount
FROM Production.ProductInventory
GROUP BY Shelf
ORDER BY Shelf
```

这个查询返回如下的（部分）结果：

```
Shelf       ProductCount
A           81
B           36
C           55
D           50
E           85
F           59
...
S           17
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
T       28
U       38
V       7
W       14
Y       2
```

如果在COUNT函数圆括号内包含DISTINCT关键字，会得到那列的不重复值的总和。例如：

```
SELECT COUNT(DISTINCT Shelf) ShelfCount
FROM Production.ProductInventory
```

这个查询返回：

```
ShelfCount
21
```

解析

在本技巧的第一个示例中，计算了每个货架上的货物数。COUNT是唯一一个不忽略NULL值的聚合函数，如果ProductID是NULL的话，它还是会包含在总数中。第二个示例演示了计算Production.ProductInventory表中不重复的货架总数。如果你需要计算一个比整数数据类型所能存放的更大的值，可以使用COUNT_BIG聚合函数，它返回bigint数据类型的值。

8.1.3　找出表达式中的最小值和最大值

MAX聚合函数返回一个组中非NULL值的最大值，而MIN聚合函数返回最小值。MIN和MAX能用于数字、
259 字符和datetime列。字符数据类型的最小值和最大值由ASCII字母排序决定。datetime值的MIN和MAX基于最早的日期和最近的日期。

在本例中，会演示如何使用MIN和MAX函数来找出Production.ProductReview表的Rating数字列中的最小值和最大值：

```
SELECT  MIN(Rating) MinRating,
        MAX(Rating) MaxRating
FROM Production.ProductReview
```

这个查询返回：

```
MinRating       MaxRating
2               5
```

解析

这个技巧演示获取Production.ProductReview表中Rating列的最小值和最大值。和其他聚合函数一样，如果在SELECT子句中引用非聚合列，那么它们必须包含在GROUP BY子句中。

8.1.4　返回值的和

SUM聚合函数返回表达式中非NULL值的总和。这个示例演示了如何使用SUM聚合函数来返回每个AccountNumber中TotalDue列值的总和：

```
SELECT  AccountNumber,
        SUM(TotalDue) TotalDueBySalesOrderID
FROM Sales.SalesOrderHeader
GROUP BY AccountNumber
ORDER BY AccountNumber
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回下面的部分结果：

```
AccountNumber     TotalDueBySalesOrderID
10-4020-000001    113098.7351
10-4020-000002    32733.9695
10-4020-000003    479506.3256
...
```

解析

在这个技巧中，根据AccountNumber来计算TotalDue列的和。因为AccountNumber本身不参与聚合，它包含在GROUP BY子句中。它也包含在ORDER BY子句中以根据分组的结果进行排序。

8.1.5 使用统计聚合函数

在这个技巧中，会演示使用统计函数VAR、VARP、STDEV和STDEVP。

VAR函数根据数据母体的样本返回由表达式提供的值的方差(VARP函数也返回表达式整个数据母体提供的值的方差)。

260

第一个示例返回Sales.SalesOrderHeader表中所有行的TaxAmt值的方差：

```
SELECT  VAR(TaxAmt) Variance_Sample,
      VARP(TaxAmt) Variance_EntirePopulation
FROM Sales.SalesOrderHeader
```

这个查询返回：

```
Variance_Sample          Variance_EntirePopulation
1177342.57277401         1177305.15524429
```


STDEV函数根据数据母体的样本返回由表达式提供的所有值的标准差。STDEVP函数也返回由表达式提供的所有值的标准差，只是它计算整个数据母体。这个示例返回Sales.SalesOrderDetail表中所有行的UnitPrice值的统计标准差：

```
SELECT  STDEV(UnitPrice) StandDevUnitPrice,
      STDEVP(UnitPrice)StandDevPopUnitPrice
FROM Sales.SalesOrderDetail
```

这个查询返回：

```
StandDevUnitPrice        StandDevPopUnitPrice
751.885080772954         751.881981921885
```

解析

尽管统计函数各不相同，但是实现还是相近的。特别是在本例中，每个函数都接受一个值表达式，使用表的一个列名。然后函数使用SELECT子句中指定的列对一组数据（0行或多行）进行操作，并返回一个值。

8.2 数学函数

SQL Server提供了几个能用于T-SQL代码的数学函数，如表8-2描述的那样。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表8-2　数学函数

函　数	描　述
ABS	计算绝对值
ACOS	计算其余弦是所指定参数的角（弧度）
ASIN	计算其正弦是所指定参数的角（弧度）
ATAN	计算其正切是所指定参数的角（弧度）
ATN2	计算其正切介于两个浮点表达式之间的角（弧度）
CEILING	计算大于或等于指定参数的最小整数
COS	计算余弦
COT	计算余切
DEGREES	把弧度转化为度
EXP	计算指定参数的指数值
FLOOR	返回小于或等于指定参数的最大整数
LOG	计算自然对数
LOG10	计算以10为底的对数
PI	返回PI常量
POWER	返回第一个参数对第二个参数的幂的值
RADIANS	把度转化为弧度
RAND	生成0到1之间的随机float类型值
ROUND	把指定参数的值舍入到指定精度
SIGN	如果参数是负数返回-1，如果是0返回0，如果是正数返回1
SIN	计算指定角（弧度）的正弦值
SQUARE	计算指定表达式的平方
SQRT	计算平方根
TAN	计算正切

261

下面的技巧会以示例来演示数学函数。

数学运算

这个技巧会演示4个不同的数学函数，包括POWER、SQRT、ROUND以及RAND。

第一个示例计算10的2次方：

```
SELECT POWER(10,2) Result
```

这个查询返回：

```
Result
100
```

下面的示例计算了100的平方根：

```
SELECT SQRT(100) Result
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回：

```
Result
10
```

这个示例四舍五入一个数字到小数点后的第三位：

```
SELECT ROUND(3.22245, 3) RoundedNumber
```

262

这个查询返回：

```
RoundedNumber
3.22200
```

这个示例返回一个0到1之间的随机浮点值（你的结果会与书中的不同）：

```
SELECT RAND() RandomNumber
```

这个查询返回：

```
RandomNumber
0.497749897248417
```

本技巧的最后一个示例根据提供的整数值返回一个固定的、浮点数据类型的值：

```
SELECT RAND(22) Result
```

这个查询返回：

```
Result
0.713983285609346
```

8

解析

在这个技巧中，演示了4个不同的数学函数，包括POWER、SQRT、ROUND以及RAND。每一个函数根据它的操作接受不同的参数。对于一些诸如RAND的数学函数，输入值是可选的。

8.3 字符串函数

下面的一组技巧会演示SQL Server的字符串函数。字符串函数对T-SQL编程非常有用，能让我们进行字符串清除、ASCII和普通字符之间的转换、模式搜索、移除尾部空格等。

表8-3列出了SQL Server中提供的各种字符串函数。

表8-3 字符串函数

函 数 名	描 述
ASCII和CHAR	ASCII函数接受字符表达式最左边的字符并返回ASCII码。CHAR函数把ASCII码的整数值转化为字符值
CHARINDEX和PATINDEX	CHARINDEX函数用于返回一个字符串在另外一个字符串中的起始位置。PATINDEX函数和CHARINDEX相似，只是PATINDEX允许在指定要搜索的字符串的时候使用通配符
DIFFERENCE和SOUNDEX	DIFFERENCE和SOUNDEX都用于计算发音相似的字符串。SOUNDEX为字符串分配一个4位数字码，DIFFERENCE计算两个字符串的SOUNDEX输出的相似级别
LEFT和RIGHT	LEFT函数返回字符串从左起指定个字符数的一部分字符串。RIGHT函数和LEFT函数相似，只是返回字符串从右起指定个字符数的一部分字符串

263

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

函数名	描述
LEN和DATALENGTH	LEN函数返回字符串表达式的字符个数，不包括最后一个字符后面的任何空格（尾部空格）。而DATALENGTH返回表达式使用的字节数
LOWER和UPPER	LOWER函数返回字符表达式的小写形式，UPPER函数返回字符表达式的大写形式
LTRIM和RTRIM	LTRIM函数移除前导空格，RTRIM函数移除尾部空格
NCHAR和UNICODE	UNICODE函数返回字符表达式或输入表达式的第一个字符的Unicode整数值。NCHAR函数接受一个表示Unicode字符的整数值并把它转化为等价的字符
QUOTENAME	QUOTENAME函数为Unicode输入字符串增加分隔符，使之成为一个有效的分隔标识符
REPLACE	REPLACE函数用一个新的字符串替换某个字符串中的一个指定字符串的所有示例
REPLICATE	REPLICATE函数将某个字符表达式重复指定次
REVERSE	REVERSE函数接受一个字符表达式，并且以逆序的字符位置输出表达式
SPACE	SPACE函数根据为输入参数指定的整数值返回一个由重复空格组成的字符串
STR	STR函数把数字数据转化为字符数据
STUFF	STUFF函数删除指定长度的字符，并在指定的起始位置插入指定字符串
SUBSTRING	SUBSTRING函数返回某个表达式中定义的一块

接下来的几个技巧会演示如何使用这些字符串函数。

8.3.1　把字符值转化为ASCII以及把ASCII转回字符

ASCII函数接受字符表达式最左边的字符并返回ASCII码，而CHAR函数将代表ASCII的整数值转化为字符值。再次强调，ASCII只使用字符串的第一个字符。如果字符串是空或NULL，ASCII会返回NULL值（但单个空格值返回32）。

264

第一个示例演示如何把字符转化为整数的ASCII值：

```
SELECT ASCII('H'), ASCII('e'), ASCII('l'), ASCII('l'), ASCII('o')
```

这个查询返回：

```
72    101   108   108   111
```

接着，使用CHAR函数来把整数值再转化回字符：

```
SELECT CHAR(72), CHAR(101), CHAR(108), CHAR(108), CHAR(111)
```

这个查询返回：

```
H     e     l     l     o
```

解析

在这个技巧中，把Hello这个单词拆成一个一个字符，然后使用ASCII函数把它转化成数字的ASCII值。在第二个T-SQL语句中，又使用CHAR函数把ASCII值转回字符。

8.3.2　返回整数和字符Unicode值

UNICODE函数返回字符或输入表达式的第一个字符的Unicode整数值。NCHAR函数接受一个表示Unicode字符的整数值，然后把它转化为等价的字符。如果你需要使用Unicode标准与外部处理程序交

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

换数据，那么这些函数就很有用。

第一个示例把单个字符转化为表示Unicode标准字符代码的整数值：

```
SELECT UNICODE('G'), UNICODE('o'), UNICODE('o'), UNICODE('d'), UNICODE('!')
```

这个查询返回：

```
71    111   111   100   33
```

接着，把Unicode整数值转回字符：

```
SELECT NCHAR(71), NCHAR(111), NCHAR(111), NCHAR(100), NCHAR(33)
```

这个查询返回：

```
G     o     o     d     !
```

解析

在这个技巧中，把单词“Good!”拆分成一个一个字符，然后使用UNICODE函数把它转化成整数值。在第二个T-SQL语句中，使用NCHAR函数把整数值转回字符。

265

8.3.3 获取某个字符串在另一个字符串中的起始位置

CHARINDEX函数用于返回某个字符串在另外一个字符串中的起始位置。语法如下：

```
CHARINDEX ( expression1 ,expression2 [ , start_location ] )
```

expression1参数是要寻找的字符串。expression2参数是要在其中搜索的字符串。可选的start_location值指示你希望开始查找的字符位置。

这个示例演示了如何查找某个字符串在另外一个字符串中的起始位置：

```
SELECT CHARINDEX('String to Find',
    'This is the bigger string to find something in.')
```

这个查询返回：

```
20
```

8.3.4 使用通配符找到某个字符串在另一个字符串中的起始位置

PATINDEX函数和CHARINDEX类似，只是PATINDEX允许在要搜索的字符串中使用通配符。PATINDEX语法如下：

```
PATINDEX ( '%pattern%' ,expression )
```

PATINDEX返回搜索模式第一次出现的起始位置，但是和CHARINDEX不同，它没有起始位置选项。在本例中，返回Person.Address表中AddressLine1包含单词Olive且地址以35开头的行：

```
SELECT  AddressID,
      AddressLine1
FROM Person.Address
WHERE PATINDEX('[3][5]%olive%', AddressLine1) > 0
```

这个查询返回下面部分结果：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
AddressID    AddressLine1
26857        3507 Olive Dr.
12416        3507 Olive Dr.
12023        3538 Olivewood Ct.
27157        3538 Olivewood Ct.

(4行受影响)
```

266

解析

本例返回所有AddressLine1列包含单词Olive并且以35开头的街道号码为前缀(使用通配符模式[3]和[5])的行。在单词左右都有通配符(之间没有空格)，这里的Olive单词还能嵌入在其他单词中。这个模式还能使用不同的通配符。

8.3.5　检测字符串相似度

DIFFERENCE和SOUNDEX两个函数都是和字符串配合使用，用于根据英语发音规则来评价字符串的发音相似度。SOUNDEX为字符串分配4个数字的代码，然后DIFFERENCE计算两个字符串的SOUNDEX输出之间的相似级别。DIFFERENCE返回0到4的值，4表示最相近的匹配。

本例演示了如何识别发音相似的字符串——首先独立运算字符串，然后成对进行比较：

```
SELECT  SOUNDEX('Fleas'),
        SOUNDEX('Fleece'),
        SOUNDEX('Peace'),
        SOUNDEX('Peas')
```

这个查询返回：

```
F420  F420  P200  P200
```

然后使用DIFFERENCE比较字符串对：

```
SELECT DIFFERENCE ( 'Fleas', 'Fleece')
```

这个查询返回：

```
4
```

然后，比较另外一对：

```
SELECT DIFFERENCE ( 'Fleece', 'Peace')
```

这个查询返回：

```
2
```

解析

在第一个示例中，使用SOUNDEX来评价4个发音相似的单词。查询返回了一个4位代码，Fleas和Fleece等于F420，而Peace和Peas等于P200。在第二个示例中，用DIFFERENCE来评价Fleas和Fleece以及Fleece和Peace。结果得出Fleas和Fleece的值为4，Fleece和Peace的值为2，前者比后者更相似。

267

8.3.6　获取字符串最左和最右部分

LEFT函数返回字符串左起指定字符个数的一部分。RIGHT函数和LEFT函数相似，只是它返回字符串

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

右起指定字符个数的一部分。

这个技巧演示如何返回字符串最左和最右部分的子集。而且，还演示了使用这些函数进行的常见的字符串填充技巧。

在第一个示例中，从字符串获取最左边的10个字符：

```
SELECT LEFT('I only want the leftmost 10 characters.', 10)
```

这个查询返回：

```
I only wan
```

接着，获取字符串最右边的10个字符：

```
SELECT RIGHT('I only want the rightmost 10 characters.', 10)
```

这个查询返回：

```
haracters.
```

下面的示例演示使用0填充ListPrice列的值：

```
-- 为了业务目的需要填充数字
SELECT TOP 3
      ProductID, RIGHT('0000000000' + CONVERT(varchar(10), ListPrice),10)
FROM  Production.Product
WHERE ListPrice > 0
```

这个查询返回：

```
ProductID     (No column name)
514           0000133.34
515           0000147.14
516           0000196.92
```

解析

这个技巧演示了使用LEFT和RIGHT的3个示例。前两个示例演示了返回字符串值的最左边和最右边的一些字符。第三个示例演示了填充字符串以符合某个期望的业务格式。如果要把数据展现给终端用户或者把数据导出到外部系统，有的时候你可能需要保留或者增加前导值，比如增加前导0来得到固定长度的数字或为varchar字段增加空格。要为ListPrice填充0，首先需要将字符串中的10个0拼接到ListPrice中转换后的varchar(10)中。然后，在这个拼接的外面，使用RIGHT来提取拼接后的字符串中最后10个字符（因此，如果ListPrice不到10个数字，在左边就填充了前导的0）。

268

8.3.7 检测字符串中的字符数或者字节数

LEN函数返回字符串表达式中的字符个数，不包含最后一个字符后的任何空格（尾部空格）。而DATALENGTH返回表达式使用的字节数。在本技巧中，会演示如何测量字符串中的字符数和字节数。

第一个示例返回Unicode字符串中的字符数（Unicode数据的每个字符占两个字节，而非Unicode数据的字符只占一个字节）：

```
SELECT LEN(N'She sells sea shells by the sea shore.')
```

这个查询返回：

```
38
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下面的示例返回Unicode字符串中的字节数。

```
SELECT DATALENGTH(N'She sells sea shells by the sea shore.')
```

这个查询返回：

```
76
```

解析

这个技巧使用了Unicode字符串，它通过给字符串加前缀N来定义，如下所示：

```
N'She sells sea shells by the sea shore.'
```

根据LEN，字符串中的字符数为38，然而由于它是Unicode字符串，DATALENGTH返回76个字节。SQL Server使用Unicode UCS-2编码形式，每个字符占2个字节。

8.3.8　把字符串的一部分替换成另一个字符串

REPLACE函数将某个字符串中的指定字符串的所有实例替换为一个新的字符串。REPLACE的一个好处在于它不像PATINDEX和CHARINDEX只返回找到模式的某个位置，REPLACE能找到某个字符串中的一个模式的多个实例。

REPLACE的语法如下：

```
REPLACE ( 'string_expression1' , 'string_expression2' , 'string_expression3' )
```

第一个字符串表达式参数是需要修改的字符串。第二个字符串表达式是需要从第一个字符串参数中移除的字符串。第三个字符串表达式是要插入到第一个参数中的字符串。

这个示例演示了如何将指定字符串的所有实例替换为一个新串：

269

```
SELECT REPLACE('Zenon is our major profit center. Zenon leads the way.',
               'Zenon',
               'Xerxes')
```

这个查询返回：

```
Xerxes is our major profit center. Xerxes leads the way.
```

解析

在这个技巧中，第一个字符串表达式是需要搜索的字符串“Zenon is our major profit center. Zenon leads the way”。第二个表达式是要替换的表达式（Zenon）。第三个表达式是替代Zenon的值Xerxes。我使用了REPLACE函数用Xerxex替换所有出现的Zenon。尽管Zenon在原字符串中出现过两次，REPLACE用一次函数调用就用Xerxes完成了两次替换。

8.3.9　把字符串填充到字符串中

STUFF函数删除某个长度的字符，并在指定的起始位置插入指定的字符串。语法如下：

```
STUFF ( character_expression, start, length, character_expression )
```

这个函数的第一个参数是要修改的字符表达式。第二个参数是要插入字符串的起始位置。length是要在字符表达式中删除的字符数。第四个参数是你希望插入的实际字符表达式。

本例替换了字符串的一部分，并在字符串内插入一个新的表达式：

```
SELECT STUFF ( 'My cat''s name is X. Have you met him?',
               18,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        1,
        'Edgar' )
```

这个查询返回：

```
My cat's name is Edgar. Have you met him?
```

解析

本技巧中的字符表达式是“My cat's name is X. Have you met him?”。起始值是18，也就是说替换会发生在字符串中的第18个位置（在这里是X）。长度值是1，也就是从位置18开始的1个字符会被删除。最后一个字符表达式是Edgar，它就是要填充到字符串中的值。

8.3.10 在小写和大写之间进行转化

LOWER函数返回字符表达式的小写形式，UPPER函数返回字符表达式的大写形式。

提示 没有内建的合适大小写的函数，所以在第11章会演示一个可以完成这个操作的标量用户定义函数。 270

在看这些不同的函数之前，我写了一个查询来显示Production.Document表中某行的DocumentSummary值：

```
SELECT DocumentSummary
FROM Production.Document
WHERE FileName = 'Installing Replacement Pedals.doc'
```

这个查询返回下面的句子大小写值：

```
DocumentSummary
Detailed instructions for replacing pedals with Adventure Works Cycles replacement
pedals.  Instructions are applicable to all Adventure Works Cycles bicycle models
and replacement pedals. Use only Adventure Works Cycles parts when replacing worn or
broken
```

第一个示例演示了把值设置为小写：

```
SELECT LOWER(DocumentSummary)
FROM Production.Document
WHERE FileName = 'Installing Replacement Pedals.doc'
```

这个查询返回：

```
detailed instructions for replacing pedals with adventure works cycles replacement
pedals.  instructions are applicable to all adventure works cycles bicycle models
and replacement pedals. use only adventure works cycles parts when replacing worn or
broken
```

现在是大写形式：

```
SELECT UPPER(DocumentSummary)
FROM Production.Document
WHERE FileName = 'Installing Replacement Pedals.doc'
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DETAILED INSTRUCTIONS FOR REPLACING PEDALS WITH ADVENTURE WORKS CYCLES REPLACEMENT
PEDALS.  INSTRUCTIONS ARE APPLICABLE TO ALL ADVENTURE WORKS CYCLES BICYCLE MODELS
AND REPLACEMENT PEDALS. USE ONLY ADVENTURE WORKS CYCLES PARTS WHEN REPLACING WORN OR
BROKEN
```

解析

第一个示例演示了LOWER函数，它返回字符表达式的小写形式。第二个示例演示了UPPER函数，它返回了字符表达式的大写形式。在这两个例子中，函数都只有一个参数，即要转化大小写的字符表达式。

8.3.11　移除前导空格和尾部空格

LTRIM函数移除前导空格，而RTRIM函数移除尾部空格。

第一个示例演示移除字符串的前导空格：

271

```
SELECT LTRIM('  String with leading blanks.')
```

这个查询返回：

```
String with leading blanks.
```

第二个示例演示移除字符串的尾部空格：

```
SELECT RTRIM('"' + 'String with trailing blanks           ') + '"'
```

这个查询返回：

```
"String with trailing blanks"
```

解析

LTRIM和RTRIM都接受一个参数——要截断的字符表达式。然后函数截断前导或尾部的空格。我们注意到，没有可以用于移除前导和尾部空格的TRIM函数（在其他编程语言中可以发现）。我们必须通过在一个表达式中使用LTRIM和RTRIM来实现。

8.3.12　重复一个表达式 N 次

REPLICATE函数将某个字符表达式重复指定的次数。

语法如下：

```
REPLICATE ( character_expression ,integer_expression )
```

第一个参数是要重复的字符表达式。第二个参数是字符表达式要重复的次数（整数值）。

这个示例演示了如何使用REPLICATE函数来重复字符表达式几次：

```
SELECT REPLICATE ('Z', 30)
```

这个查询返回：

```
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

解析

在这个技巧的示例中，字符表达式中的字母Z重复了30次。使用REPLICATE就可以不用手动编码这些字符来重复这些值。最大返回值是8 000字节。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

8.3.13 重复一个空格N次

SPACE函数根据为输入参数指定的整数，返回由重复空格组成的字符串。这个函数与REPLCATE函数功能相似——只是它用在特定的字符常量上。

本例演示了如何将空格重复指定次数：

```
SELECT 'Give me some' + SPACE(6) + 'space.'
```

272

这个查询返回：

```
Give me some      space.
```

解析

在这个技巧中，我们使用SPACE函数在字符串的中间拼接了6个空格。最大返回值是8 000个字节。

8.3.14 逆序输出表达式

REVERSE函数接受一个字符表达式，并以逆序的字符位置输出表达式。

本例演示了如何反转一个字符串表达式：

```
SELECT TOP 1
      GroupName,
      REVERSE(GroupName) GroupNameReversed
FROM HumanResources.Department
ORDER BY GroupName
```

这个查询返回：

```
GroupName                                GroupNameReversed
Executive General and Administration     noitartsinimdA dna lareneG evitucexE
```

解析

这个技巧演示了使用REVERSE函数以逆序输出字符串的字符。

8.3.15 返回表达式的一块

SUBSTRING函数返回某个表达式的指定块。

语法如下：

```
SUBSTRING ( expression, start, length )
```

函数的第一个参数是你希望使用其来返回指定块的字符表达式。第二个参数定义了块的起始位置，第三个参数是你希望提取的字符块的长度。

在本例中，假设你的应用程序接收到了一个客户的银行账号。按照公司的规定只能以隐藏形式保存银行账号，只保留中间的4位：

```
DECLARE @BankAccountNumber char(14)
SET @BankAccountNumber = '1424-2342-3536'

SELECT 'XXXX-' + SUBSTRING(@BankAccountNumber, 6,4) + '-XXXX'
 Masked_BankAccountNumber
```

273

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Masked_BankAccountNumber
XXXX-2342-XXXX
```

解析

在这个技巧中，我们使用SUBSTRING函数来获取长串银行账号的中间4位。第一个参数是要从中获取中间四个字符的银行账号。第二个参数要获取字符串的开始位置，在这里是第六个位置（即第一个2），第三个参数表示要获取多少个字符（在这个例子中为4）。结果是从银行账号中获取的值2342，并且插入到掩码字符串中。

8.4　处理 NULL

由于NULL值是未知的，它可能会弄乱相关代码。SQL Server提供了函数来处理代码中的NULL值，如表8-4描述的那样。

表8-4　NULL函数

函 数 名	描　述
ISNULL	ISNULL检测表达式是否为NULL，如果是的话替换NULL值为另外一个值
COALESCE	COALESCE函数返回指定表达式列表的第一个非NULL值
NULLIF	当指定的两个表达式有相同值的时候，NULLIF返回NULL值，否则返回第一个表达式的值

下面的几个技巧会演示这些函数。

8.4.1　用另一个值替换 NULL 值

ISNULL验证表达式是否为NULL，如果是的话，用另一个值替换NULL值。

在本例中，任何NULL值都会被替换为另外的值：

```
SELECT JobCandidateID,
       BusinessEntityID,
       ISNULL(BusinessEntityID, 0) Cleaned_BusinessEntityID
FROM HumanResources.JobCandidate
```

这个查询返回下面的（部分）结果：

```
JobCandidateID      BusinessEntityID    Cleaned_BusinessEntityID
1                   NULL                0
2                   NULL                0
...
13                  NULL                0
8                   212                 212
4                   274                 274
```

274

解析

在这个示例中，一些行中的BusinessEntityID列包含NULL值。我在查询的第二列显示了原始行值。在查询的第三列，我用ISNULL函数包装了BusinessEntityID列。在函数的第二个参数中，我指定用0替换NULL值。

8.4.2　使用 ISNULL 进行灵活的搜索

在这个技巧中，会演示如何在变量可能有或没有填充的情况下在查询中进行灵活、动态的搜索。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个技巧声明了3个局部搜索变量ProductID、StartDate和StandardCost。通过使用这个技术，查询可以根据所有、一些或者0个参数来返回结果。在这个示例中，只指定了ProductID：

```
-- 用于搜索的局部变量
DECLARE @ProductID int
DECLARE @StartDate datetime
DECLARE @StandardCost money

-- 只使用了@ProductID
SET @ProductID = 711

SELECT ProductID, StartDate, StandardCost
FROM Production.ProductCostHistory
WHERE ProductID = ISNULL(@ProductID, ProductID) AND
     StartDate = ISNULL(@StartDate, StartDate) AND
     StandardCost = ISNULL(@StandardCost, StandardCost)
```

这个查询返回：

```
ProductID     StartDate                StandardCost
711           2001-07-01 00:00:00.000  12.0278
711           2002-07-01 00:00:00.000  13.8782
711           2003-07-01 00:00:00.000  13.0863
```

在第二个示例中，通过最小和最大的StandardCost范围来进行搜索：

```
-- 用于搜索的局部变量
DECLARE @ProductID int
DECLARE @MinStandardCost money
DECLARE @MaxStandardCost money

SET @MinStandardCost = 3.3963
SET @MaxStandardCost = 10.0000

SELECT ProductID, StartDate, StandardCost
FROM Production.ProductCostHistory
WHERE ProductID = ISNULL(@ProductID, ProductID) AND
 StandardCost BETWEEN ISNULL(@MinStandardCost, StandardCost) AND
ISNULL(@MaxStandardCost, StandardCost)
ORDER BY StandardCost
```

这个查询返回下面的（部分)结果：

```
ProductID     StartDate                StandardCost
709           2001-07-01 00:00:00.000  3.3963
710           2001-07-01 00:00:00.000  3.3963
871           2003-07-01 00:00:00.000  3.7363
712           2002-07-01 00:00:00.000  5.2297
...
932           2003-07-01 00:00:00.000  9.3463
860           2002-07-01 00:00:00.000  9.7136
859           2002-07-01 00:00:00.000  9.7136
858           2002-07-01 00:00:00.000  9.7136
```

解析

本技巧中演示的方法的优势在于代码很灵活，可以以各种方式搜索数据，并且能让每个搜索条件为可选的。这个技巧的关键在于WHERE子句。每一个搜索条件都使用ISNULL和局部变量名，后面跟列名

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

本身：

```
WHERE ProductID = ISNULL(@ProductID, ProductID) AND
      StartDate = ISNULL(@StartDate, StartDate) AND
      StandardCost = ISNULL(@StandardCost, StandardCost)
```

如果参数没有设置，它就会是NULL，因此每列的搜索条件会对列本身的值进行运算——总是返回TRUE。只有指定的参数才会用来过滤结果。

8.4.3　返回表达式列表中第一个非 NULL 值

COALESCE函数从提供的表达式列表中返回第一个非NULL值。语法是：

```
COALESCE ( expression [ ,...n ] )
```

这个技巧演示了如何使用COALESCE来返回第一个出现的非NULL值：

```
DECLARE @Value1 int
DECLARE @Value2 int
DECLARE @Value3 int

SET @Value2 = 22
SET @Value3 = 955

SELECT COALESCE(@Value1, @Value2, @Value3)
```

这个查询返回：

```
22
```

276

解析

在这个技巧中，我们创建了3个局部变量：@Value1、@Value2和@Value3。只有@Value2和@Value3被设置为整数值。没有设置值的变量@Value1是NULL。在COALESCE中，从@Value1到@Value3一次检查3个值。由于@Value2变量是第一个非NULL值的变量，所以返回了22。

8.4.4　当两个表达式相等的时候返回 NULL 值，否则返回第一个表达式

当提供的两个表达式有相同值的时候，NULLIF返回NULL值；否则返回第一个表达式。

本例演示了如何使用NULLIF来运算两个表达式。如果两个表达式相等，NULL值会被返回；否则会返回第一个运算的表达式：

```
DECLARE @Value1 int
DECLARE @Value2 int

SET @Value1 = 55
SET @Value2 = 955

SELECT NULLIF(@Value1, @Value2)
```

这个查询返回：

```
55
```

下面的示例测试两个表达式相等时返回的值：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DECLARE @Value1 int
DECLARE @Value2 int

SET @Value1 = 55
SET @Value2 = 55

SELECT NULLIF(@Value1, @Value2)
```

这个查询返回：

```
NULL
```

解析

在这个技巧中，开始的批处理有两个不同的值：55和955。由于值不同，因此NULLIF条件为FALSE，并返回第一个运算的值。在第二个批处理中，@Value1和@Value2相等，所以NULLIF返回了NULL值。

8.5 日期函数

在本书的开始我们已经提到过，SQL Server有几种用来存储日期和时间数据的数据类型：datetime、datetime2、date、time、datetimeoffset和smalldatetime。SQL Server提供了一些使用这些数据类型并进行操作的函数，如表8-5所示。

277

表8-5 日期函数

函 数 名	描 述
DATEADD	DATEADD根据指定的间隔和数字返回一个增加或减少后的新日期
DATEDIFF	DATEDIFF从第二个日期减去第一个日期，并生成一个指定的datepart码格式的值
DATENAME	DATENAME返回datepart码指定的一部分日期的字符串值
DATEPART	这个函数返回datepart码指定的一部分日期的整数值
DAY、MONTH和YEAR	DAY返回表示日的整数值，MONTH返回表示月的整数，而YEAR返回表示计算日期的年的整数
GETDATE、GETUTCDATE和CURRENT_TIMESTAMP	GETDATE和CURRENT_TIMESTAMP都返回当前的日期和时间。GETUTCDATE返回格林尼治平时（UTC）
ISDATE	当表示式为有效日期或时间时，ISDATE返回1（true），否则返回0（false）
SYSDATETIME、SYSUTCDATETIME和SYSDATETIMEOFFSET	SYSDATETIME以datetime2的形式返回当前日期和时间。SYSUTCDATETIME以datetime2的形式返回UTC。SYSDATETIMEOFFSET返回当前日期和时间，并以datetimeoffset的形式返回当前时区和UTC之间小时分钟的偏移量。这些函以10毫秒的精度返回时间
SWITCHOFFSET	SWITCHOFFSET允许你以datetimeoffset数据类型形式将既有的时区偏移量修改为新的偏移量
TODATETIMEOFFSET	TODATETIMEOFFSET允许你将日期和时间值修改为指定的时区偏移量，并以datetimeoffset的数据类型形式返回值

下面的一些技巧会演示这些日期函数。

8.5.1 返回当前日期和时间

GETDATE和CURRENT_TIMESTAMP都返回当前的日期和时间。GETUTCDATE返回协调世界时。SQL Server

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

2008的新日期函数SYSDATETIME和SYSUTCDATETIME提供了10毫秒精度的日期和时间。SYSDATETIMEOFFSET也提供了这个级别的精度，但它还包括了同UTC的小时分钟的偏移量。

这个示例演示了如何返回当前日期和时间，还有世界协调时和相应的偏移量：

```
SET NOCOUNT ON

SELECT 'CurrDateAndTime_HighPrecision', SYSDATETIME()

SELECT 'UniversalTimeCoordinate_HighPrecision', SYSUTCDATETIME()

SELECT 'CurrDateAndTime_HighPrecision _UTC_Adjust', SYSDATETIMEOFFSET()
```

278 这个查询返回：

```
CurrDateAndTime_HighPrecision              2008-03-09 08:10:12.0861608

UniversalTimeCoordinate_HighPrecision      2008-03-09 14:10:12.0861608

CurrDateAndTime_HighPrecision _UTC_Adjust  2008-03-09 08:10:12.0861608 -06:00
```

解析

这个技巧演示了获取当前日期和时间的方法。所有这些函数都能在表的列定义中用作日期数据类型的DEFAULT值。

8.5.2　在时区间进行转换

随后的技巧演示使用函数调整并转换datetimeoffset和datetime数据类型值为datetimeoffset数据类型值。在第一个查询中，SWITCHOFFSET函数将一个既有的datetimeoffset值从–05:00转换为+03:00。

datetimeoffset数据类型的值表示以UTC时间为参考的指定时区的日期和时间。例如，U.S.东部标准时区被定义为UTC – 5:00（UTC减5小时）。你可以通过调用SWITCHOFFSET函数将datetimeoffset值在时区之间进行转换。在随后的示例中，输入值从UTC – 5:00转换为 UTC + 3:00：

```
SELECT SWITCHOFFSET ( '2007-08-12 09:43:25.9783262 -05:00', '+03:00')
```

转换的效果是更改值的datetime部分，使得新组合的datetime和偏移量表示与之前的UTC时间一致的时间。在这个例子中，结果是：

```
2007-08-12 17:43:25.9783262 +03:00
```

2007-08-12 09:43:25.9783262 -05:00（输入）和2007-08-12 17:43:25.9783262 +03:00（输出）表示同一时间，都是UTC时间2007-08-12 14:43:25.978326（偏移量0:00）。

在第二个查询中。TODATETIMEOFFSET把常规的datetime数据类型值（没有相关联的时区）转换为datetimeoffset数据类型值：

```
SELECT TODATETIMEOFFSET ( '2007-08-12 09:43:25' , '-05:00' )
```

这个查询返回：

```
2007-08-12 09:43:25.0000000 -05:00
```

解析

这个技巧使用了SQL Server 2008中引入的两个新函数，用来操作值并将值转换为datetimeoffset

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

数据类型。在第一个示例中，SWITCHOFFSET函数据使用两个输入参数：要调整的datetimeoffset值和要调整值的偏移量值。不管原始的值的偏移量是什么，都被转换为第二个参数指定的偏移量值。

279

在第二个示例中，TODATETIMEOFFSET函数也有两个输入参数：要转换的datetime值和将值转换成datetimeoffset数据类型时用到的偏移量值。

8.5.3 增加或减少日期值

DATEADD返回一个新的日期，它是增加或减少另外一个日期表达式的结果。语法如下：

```
DATEADD (datepart , number, date )
```

datepart部分用于指定要修改日期的哪一部分时间，如表8-6描述的那样。

表8-6 datepart代码

代 码	描 述
yy或者yyyy	年
qq或q	季度
mm或m	月
dy或y	年中第几天
dd或d	天
wk或ww	周
dw或w	星期几
hh	小时
mi或n	分钟
ss或s	秒
ms	毫秒

DATEADD函数的第二个参数是要增加或减少日期的数字值（正数或负数）。第三个参数是要修改的日期。

第一个示例将日期减一年：

```
SELECT DATEADD(yy, -1, '4/2/2009')
```

这个查询返回：

```
2008-04-02 00:00:00.000
```

下面的示例将日期增加一个季度：

```
SELECT DATEADD(q, 1, '4/2/2009')
```

这个查询返回：

```
2009-07-02 00:00:00.000
```

280

这个示例将一个日期减少6个月：

```
SELECT DATEADD(mm, -6, '4/2/2009')
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
2008-10-02 00:00:00.000
```

这个示例日期增加50天：

```
SELECT DATEADD(d, 50, '4/2/2009')
```

这个查询返回：

```
2009-05-22 00:00:00.000
```

这个示例将日期和时间减少30分钟：

```
SELECT DATEADD(mi, -30, '2009-09-01 23:30:00.000')
```

这个查询返回：

```
2009-09-01 23:00:00.000
```

解析

这个技巧演示了使用DATEADD函数根据不同的粒度来修改日期。对于前面的每个示例来说，DATEADD的第三个参数都是文本日期值。然而，也可以引用datetime数据类型的表列或有效的日期表达式。你会在下面的技巧中看到，第一个参数detepart也能在不同的日期函数中使用。

8.5.4　找出两个日期的差

DATEDIFF让第二个日期减去第一个日期，产生一个指定的datepart代码格式的值。DATEDIFF的语法如下：

```
DATEDIFF ( datepart , startdate , enddate )
```

第一个datepart代码使用和DATEADD相同的datepart代码。第二个和第三个参数是参与减法的日期值。

本例演示了如何使用DATEDIFF函数来找出两个日期的差：

```
-- 找出当前时间和EndDate之间的差
SELECT ProductID,
       GETDATE() Today,
       EndDate,
       DATEDIFF(m, EndDate, GETDATE()) MonthsFromNow
FROM Production.ProductCostHistory
WHERE EndDate IS NOT NULL
```

281

这个查询返回下面的（部分）结果：

```
ProductID   Today                   EndDate                 MonthsFromNow
707         2008-02-12 19:07:14.073 2002-06-30 00:00:00.000 68
707         2008-02-12 19:07:14.073 2003-06-30 00:00:00.000 56
708         2008-02-12 19:07:14.073 2002-06-30 00:00:00.000 68
708         2008-02-12 19:07:14.073 2003-06-30 00:00:00.000 56
...
```

解析

在这个技巧中，计算ProductCostHistory表的EndDate和当前日期的差，按月返回这个差。下一个技巧演示了另一个使用datepart为参数的函数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

8.5.5 显示日期一部分的字符串值

DATENAME返回以datepart代码指定的部分日期的字符串值。语法如下：

```
DATENAME ( datepart , date )
```

第二个参数指定了字符串值基于的日期。

在这个技巧中，我会演示如何使用DATENAME返回指定日期是星期几：

```
-- 显示EndDate这天为星期几
SELECT ProductID,
      EndDate,
      DATENAME(dw, EndDate) WeekDay
FROM Production.ProductCostHistory
WHERE EndDate IS NOT NULL
```

这个查询返回下面的（部分）结果：

```
ProductID    EndDate                  WeekDay
707          2002-06-30 00:00:00.000  Sunday
707          2003-06-30 00:00:00.000  Monday
708          2002-06-30 00:00:00.000  Sunday
708          2003-06-30 00:00:00.000  Monday
709          2002-06-30 00:00:00.000  Sunday
```

解析

在这个技巧中，datepart参数设置成dw（星期几），结果根据EndDate列日期返回了星期几的值。

8

8.5.6 使用 DATEPART 显示日期一部分的整数值

这个函数根据datepart指定的部分日期返回整数值。DATEPART语法如下：

282

```
DATEPART ( datepart , date )
```

第二个参数date指定了要计算整数值的日期。

这个示例演示了如何根据选择的日期部分返回整数值。第一个示例返回年值：

```
SELECT DATEPART(yy, GETDATE())
```

这个查询返回：

```
2008
```

下一个示例显示当前月份的整数值：

```
SELECT DATEPART(m, GETDATE())
```

这个查询返回：

```
2
```

解析

在这个技巧中，使用DATEPART函数从当前的日期和时间中提取年、月和日的整数值。也可以使用不需要datepart参数的封装好的函数来显示这些值，下面的技巧会讨论。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

8.5.7　使用 YEAR、MONTH 和 DAY 显示日期一部分的整数值

也可以使用单个参数的函数来显示日、月和年的整数值：

这个示例返回当前年：

```
SELECT YEAR(GETDATE())
```

这个查询返回：

```
2008
```

这个示例返回当前月：

```
SELECT MONTH(GETDATE())
```

这个查询返回：

```
9
```

这个示例返回当前日：

283

```
SELECT DAY(GETDATE())
```

这个查询返回：

```
30
```

解析

在这个技巧中，演示了单个参数的日期函数。DAY返回日的整数值，MONTH返回表示月的整数，YEAR返回表示计算日期的年的整数。

8.6　类型转换

CONVERT和CAST函数都是用来将一种数据类型转换为另一种数据类型。CAST的语法如下所示：

```
CAST ( expression AS data_type [ (length ) ])
```

第一个参数是要转换的表达式（例如一个表列或字面量值）。第二个参数是要转换成的数据类型。

CONVERT的语法如下：

```
CONVERT ( data_type [ ( length ) ] ,expression [ ,style ] )
```

第一个参数是要转换成的数据类型。第二个参数是需要转换的表达式。第三个参数style允许我们配置某个日期呈现的格式。使用CAST函数时不可以使用第三个参数。

8.6.1　转换数据类型

在这个技巧中，会演示如何将整数数据类型转换为char(4)数据类型。

在第一个示例中，把整数与字符串拼接：

```
SELECT 2000 + 'Cannot be concatenated'
GO
```

这个查询返回下面的错误：

```
消息245，级别16，状态1，第1行
在将varchar值'Cannot be concatenated'转换成数据类型int时失败。
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在下面的示例中，使用CONVERT来将整数值变为字符数据类型：

```
SELECT CONVERT(char(4), 2008) + ' Can now be concatenated!'
```

这个查询返回：

```
2008 Can now be concatenated!
```

284

这个示例演示了使用CAST进行相同类型的转换：

```
SELECT BusinessEntityID, CAST(SickLeaveHours AS char(4)) +
       ' Sick Leave Hours Left' SickTime
FROM HumanResources.Employee
```

这个查询返回下面的（部分）结果：

```
BusinessEntityID      SickTime
1                     30  Sick Leave Hours Left
2                     41  Sick Leave Hours Left
3                     21  Sick Leave Hours Left
4                     80  Sick Leave Hours Left
5                     24  Sick Leave Hours Left
```

解析

第一个查询试图把整数和字符串值拼接在一起。这将会出现一个错误，因为两个数据类型必须是兼容的或者是一致的。第二次尝试使用CONVERT修改表达式的数据类型为char(4)，然后将表达式和其他字符串拼接起来。CAST用来转换smallint列的数据类型，这样就能把它与字符串进行拼接。

8.6.2 将日期转换为文本形式

前面提到过，CONVERT有一个可选的样式参数，可以用它将datetime或smalldatetime转换为指定的字符格式。许多人很疑惑日期和时间是如何存储，以和查询结果中实际的日期表示相符的。使用样式参数的时候，请记住你所做的只会影响日期的字符表现方式，而不会影响它的存储方式（当然，除非使用非datetime数据类型列存储要表现的日期）。

表8-7中列出了CONVERT函数可用样式格式的一些示例。

表8-7 CONVERT样式格式

样式代码	格 式
101	mm/dd/yyyy
102	yy.mm.dd
103	dd/mm/yy
108	hh:mm:ss
110	mm-dd-yy
112	yymmdd

例如，

```
SELECT CONVERT(varchar(20), GETDATE(), 101)
```

返回的当天日期格式为：

```
02/12/2008
```

285

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

当诸如GETDATE()的函数执行并保存在datetime列的时候，指定的日期和时间数据都会保存。然而，如果仅仅希望保存日期级别的数据（不存储指定的时间），常用的技巧是使用CONVERT来消除样式为00:00:00.000的时间。

下面的示例转换datetime值为字符值，然后重新把它转换为datetime数据类型：

```
SELECT CONVERT(datetime, CONVERT( varchar(11), '2008-08-13 20:37:22.570', 101))
```

这个查询返回：

```
2008-08-13 00:00:00.000
```

当然，现在在SQL Server 2008中，不需要存储时间时，你完全可以进行如下从datetime到date数据类型的转换：

```
SELECT CONVERT(date,'2008-08-13 20:37:22.570')
```

这个查询返回：

```
2008-08-13 00:00:00.000
```

解析

在技巧的第一个查询中，在CONVERT的样式选项中使用了101值，以mm/dd/yyyy格式返回日期。在把数据呈现给终端用户的时候，查询编写者通常会关心样式选项。当要把datetime或smalldatetime转换为字符数据类型的时候就要使用这种呈现方式。记住，如果你将日期类型转换回datetime并保存这种转换后的日期，则根据你为字符数据选择的样式，你可能会丢失原始小时、分钟、秒等的精度。

在第二个查询中，使用带有101样式选项的CONVERT演示从datetime值消除时间，并将它设为00:00:00.000值。在本技巧的最后一个查询中，演示了另一个通过将datetime值转换成date数据类型来消除时间值的示例。

8.6.3　以字符串字面量表示二进制数据

SQL Server 2008引入了以字符串十六进制字面量格式返回二进制数据的新方法。CONVERT命令允许3个二进制样式：0、1和2。我将演示每个值的使用方法。

二进制样式0将二进制字节转换为ASCII字符，将ASCII字符转换为二进制字节。这是SQL Server早期版本的行为。在这个查询中，将演示二进制样式0：

```
SELECT CONVERT(char(29),
       0x53514C2053657276657220323030382042D53514C2052656369706573,
       0) ReturnValue
```

这个查询返回：

```
ReturnValue
SQL Server 2008 T-SQL Recipes
```

286

然后，通过将ASCII转换为varbinary将之前的示例反转：

```
SELECT CONVERT(varbinary, 'SQL Server 2008 T-SQL Recipes', 0) ReturnValue
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ReturnValue
0x53514C2053657276657220323030382054 2D53514C2052656369706573
```

使用SQL Server 2008带来的新功能，二进制样式1和2用来将二进制字节转化为表示十六进制值的字符串表达式。样式1是以0x为前缀的值，样式2无此前缀。如下的查询演示了样式1：

```
SELECT CONVERT(char(60),
       0x53514C2053657276657220323030382054 2D53514C2052656369706573,
       1) ReturnValue_Style_1
```

这个查询返回：

```
ReturnValue_Style_1
0x53514C2053657276657220323030382054 2D53514C2052656369706573
```

下一个查询演示了样式2：

```
SELECT CONVERT(char(60),
       0x53514C2053657276657220323030382054 2D53514C2052656369706573,
       2) ReturnValue_Style_2
```

这个查询返回如下内容（注意这里没有0x前缀）：

```
ReturnValue_Style_2
53514C2053657276657220323030382054 2D53514C2052656369706573
```

也可以将字符表达式转换为二进制，但是在这里，样式1和样式2都返回带有0x前缀（数据类型本身的）的varbinary数据类型数据。

解析

SQL Server 2008引入了将二进制数据转换为字符串十六进制字面量格式的能力。在早期的版本中，使用CONVERT会将二进制数据转换为ASCII格式——不是实际的十六进制字面量格式的表现形式。要完成这个早期版本的行为则使用二进制样式0。另外，也可以使用样式1和样式2来保持字符串十六进制字面量格式。

8.6.4 计算表达式返回的数据类型

当转换数据类型的时候，有时候知道SQL Server认为一个表达式是什么数据类型是很有用的。在这个技巧中，会演示使用ISDATE和ISNUMERIC函数来检测表达式的数据类型：

```
-- 返回0
SELECT ISDATE('1/1/20000')

-- 返回1
SELECT ISDATE('1/1/2008')

-- 返回0
SELECT ISNUMERIC('123ABC')

-- 返回1
SELECT ISNUMERIC('123')
```

这些查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
0

(1 行受影响)

1

(1 行受影响)

0

(1 行受影响)

1

(1 行受影响)
```

解析

ISDATE检测表达式是否为有效的datetime值。ISNUMERIC检测表达式是否为有效的数字数据类型值。如果表达式计算为TRUE，ISNUMERIC和ISDATE都返回1；如果为FALSE，则都返回0。

8.7　排名函数

可以使用排名函数返回和结果集中的每行关联的排名值。表8-8描述了4个排名函数。

表8-8　排名函数

函　数	描　述
ROW_NUMBER	为结果集中的每行返回递增整数
RANK	和ROW_NUMBER相似，RANK为结果集中的每行递增值。主要的区别是如果行有重复值，它们会有相同的排名值
DENSE_RANK	DENSE_RANK和RANK几乎一样，只是如果行有重复值，DENSE_RANK不在排名值中返回间隔
NTILE	NTILE根据排序和可选的分区子句把结果集划分成指定数量的分组

288　下面4个技巧会演示这4个排名函数的使用。

8.7.1　生成递增行号

ROW_NUMBER为结果集中的每行返回递增的整数。ROW_NUMBER的语法如下：

```
ROW_NUMBER ( )    OVER ( [ <partition_by_clause> ] <order_by_clause> )
```

第一个可选参数partition_by_clause允许你为分区列中的每次修改重置行号。第二个参数order_by_clause指定ROW_NUMBER应用到结果集的次序。

第一个示例返回结果集中间的6行，根据名字排序：

```
-- 选择结果集中间从255到260的行
SELECT  p.ProductID,
        p.Name,
        p.RowNumber
FROM
   (SELECT      ProductID,
          Name,
          ROW_NUMBER() OVER (ORDER BY Name) RowNumber
   FROM Production.Product) p
WHERE      p.RowNumber BETWEEN 255 AND 260
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回：

```
ProductID   Name                          RowNumber
713         Long-Sleeve Logo Jersey, S    255
716         Long-Sleeve Logo Jersey, XL   256
462         Lower Head Race               257
857         Men's Bib-Shorts, L           258
856         Men's Bib-Shorts, M           259
855         Men's Bib-Shorts, S           260
```

可选的partition_by_clause允许为分区列的每次修改重置行号。在这个示例中，结果根据Shelf来分区并且根据ProductID排序：

```
SELECT Shelf,
      ProductID,
      ROW_NUMBER() OVER
(PARTITION BY Shelf ORDER BY ProductID) RowNumber
FROM Production.ProductInventory
```

在返回的结果中，行号根据ProductID递增，但是每一次Shelf变化后，行号都会从1开始：

```
Shelf       ProductID       RowNumber
A           1               1
A           1               2
A           2               3
...
Shelf       ProductID       RowNumber
B           1               1
B           2               2
B           3               3
...
Shelf       ProductID       RowNumber
C           317             1
C           318             2
C           319             3
...
```

解析

在第一个示例中，使用ROW_NUMBER来根据产品名排列结果，并且为每行增加递增值。在子查询（从主查询中选的）的第三列中引用了ROW_NUMBER：

```
...
   SELECT     ProductID,
         Name,
         ROW_NUMBER() OVER (ORDER BY Name) RowNumber
   FROM Production.Product
...
```

括号中的ORDER BY子句根据产品名对结果排序，它影响行返回的次序以及和每行关联的行号。结果集中的每一行都会有一个号码，每行都会加1。由于查询根据Name排列结果，因此第一个产品的行号会是1。把这个查询作为子查询，这样ROW_NUMBER列就能在外部查询的WHERE子句中被引用，返回从255到260的行。

第二个示例演示了使用partition_by_clause参数。对于Shelf的每一次改变，行号都会从1开始。有了ROW_NUMBER函数，我们现在就可以不用像SQL Server 2005之前的版本那样创建其他代码实现数据分页了（例如，显示25到50行）。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

8.7.2　根据排名返回行

在这个技巧中，会演示使用RANK函数来应用基于SalesQuota值的排名值。RANK返回行在结果集中的排名（如果指定了可选的分区子句，就是行在结果集的分区中的排名）。RANK的语法如下：

```
RANK ( )  OVER ( [ < partition_by_clause > ] < order_by_clause > )
```

主要的区别是如果行有相同的值存在，那么他们会获取相同的排名值，如本例演示的那样：

```
SELECT BusinessEntityID,
       QuotaDate,
       SalesQuota,
       RANK() OVER (ORDER BY SalesQuota DESC) as RANK
FROM Sales.SalesPersonQuotaHistory
WHERE SalesQuota BETWEEN 266000.00 AND 319000.00
```

这个查询返回：

```
BusinessEntityID  QuotaDate               SalesQuota   RANK
280               2003-07-01 00:00:00.000  319000.00    1
284               2003-04-01 00:00:00.000  304000.00    2
280               2002-04-01 00:00:00.000  301000.00    3
282               2003-01-01 00:00:00.000  288000.00    4
283               2003-04-01 00:00:00.000  284000.00    5
284               2003-01-01 00:00:00.000  281000.00    6
278               2004-01-01 00:00:00.000  280000.00    7
283               2002-01-01 00:00:00.000  280000.00    7
283               2002-04-01 00:00:00.000  267000.00    9
278               2002-01-01 00:00:00.000  266000.00    10
```

290

和ROW_NUMBER一样，OVER子句包含可选的partition_by_clause和必需的order_by_clause。order_by_clause决定RANK值应用到每行的顺序，可选的partition_by_clause用于进一步划分排序分组，如下例所示：

```
SELECT h.BusinessEntityID,
   s.TerritoryID,
   h.QuotaDate,
      h.SalesQuota,
RANK() OVER (PARTITION BY s.TerritoryID ORDER BY h.SalesQuota DESC) as RANK
FROM Sales.SalesPersonQuotaHistory h
INNER JOIN Sales.SalesPerson s ON
   h.BusinessEntityID = s.BusinessEntityID
WHERE s.TerritoryID IN (5,6,7)
```

这个查询根据销售人员的TerritoryID进行分区，并返回SalesQuota的排名：

```
SalesPersonID  TerritoryID  QuotaDate               SalesQuota    RANK
279            5            2003-07-01 00:00:00.000  950000.00    1
279            5            2001-10-01 00:00:00.000  917000.00    2
...
282            6            2003-07-01 00:00:00.000  1051000.00   1
282            6            2004-04-01 00:00:00.000  830000.00    2
282            6            2001-10-01 00:00:00.000  767000.00    3
282            6            2003-10-01 00:00:00.000  707000.00    4
282            6            2002-01-01 00:00:00.000  583000.00    5
282            6            2002-04-01 00:00:00.000  583000.00    5
282            6            2004-01-01 00:00:00.000  569000.00    7
...
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

RANK根据排序的列增加它的值，和ROWNUMBER的增加每行值不同，RANK会为排序值一致的行返回相同的值。

例如，在这个技巧中，查询指定了根据SalesQuota降序排序的RANK。由于两个SalesQuota值都等于280000.00，它们都得到了排名7：

```
278 280000.00    7
283 280000.00    7
```

你也注意到了，下面的SalesQuota值排名是9（而不是8）。RANK函数没有使用第8个位置，是因为两行是并列第七，所以下一个排名值是9。如果3行并列，那么下一个排名值是10，依此类推。

```
9
```

291

在第二个示例中，RANK根据TerritoryID分区，这导致对于TerritoryID的每一次修改，RANK值都会从1开始。

8.7.3 根据无间隔排名返回行

在这个技巧中，会演示DENSE_RANK，它几乎和RANK一样，只是DENSE_RANK不返回有间隔的排名值：

```
SELECT  BusinessEntityID,
      SalesQuota,
DENSE_RANK() OVER (ORDER BY SalesQuota DESC) as DENSE_RANK
FROM Sales.SalesPersonQuotaHistory
WHERE SalesQuota BETWEEN 266000.00 AND 319000.00
```

这个查询返回：

```
BusinessEntityID    SalesQuota      DENSE_RANK
280                 319000.00       1
287                 304000.00       2
280                 301000.00       3
282                 288000.00       4
283                 284000.00       5
287                 281000.00       6
278                 280000.00       7
283                 280000.00       7
283                 267000.00       8
278                 266000.00       9
```

解析

它的语法和用法与RANK相同，只是当存在重复的记录时DENSE_RANK不在排名值中生成间隔。在这个技巧的示例中，由于两个SalesQuota是280000.00，因此两个排名都是第7：

```
278    280000.00    7
283    280000.00    7
283    267000.00    8
```

在7之后的DENSE_RANK值为8。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

8.7.4　使用 NTILE

NTILE根据排序和可选的分区把结果集分成指定数量的分组。语法和其他排名函数相似，只是它包含一个integer_expression：

```
NTILE (integer_expression) OVER ( [ < partition_by_clause > ] < order_by_clause > )
```

integer_expression用于指定要将结果划分成的分组数。本例演示对Sales.SalePersonQuotaHis-
292 tory表操作的NTILE排名函数：

```
SELECT  BusinessEntityID,
      SalesQuota,
NTILE(4) OVER (ORDER BY SalesQuota DESC) as NTILE
FROM Sales.SalesPersonQuotaHistory
WHERE SalesQuota BETWEEN 266000.00 AND 319000.00
```

这个查询返回：

BusinessEntityID	SalesQuota	NTILE
280	319000.00	1
287	304000.00	1
280	301000.00	1
282	288000.00	2
283	284000.00	2
287	281000.00	2
278	280000.00	3
283	280000.00	3
283	267000.00	4
278	266000.00	4

解析

在这个示例中，结果集被划分成4个分组。结果根据SalesQuota排序（降序），并且指定了NTILE分组分配的次序。注意到，前面的两组1和2都有3行，而分组3和4都有2行。如果行数不能被分组数整除，那么前几个分组会比后面几个分组多一些行。否则，如果行数能被分组数整除，每一个分组会有相同数量的行。

8.8　使用系统函数检测服务器、数据库以及连接级别的配置

SQL Server包含了一些用于检测SQL Server实例的系统设置的系统配置函数。有一些函数带有@@前缀，在之前版本的SQL Server中叫做变量。其他系统函数没有@@前缀，并且接受用于帮助收集有关SQL Server实例或者数据库信息的参数。

下面几个技巧会演示这些系统函数。

8.8.1　确定每周的第一天

@@DATEFIRST函数返回SQL Server实例指定的每周第一天的值。因为这个值定义了诸如DATEPART和DATEADD等其他日期函数使用的每周工作日datepart的计算，所以这是需要关注的。在本例中，会演示返回SQL Server实例当前每周第一天：

293
```
SELECT @@DATEFIRST 'First Day of the Week'
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
First Day of the Week
7
```

解析

@@DATEFIRST函数显示了每周第一天的设置。要修改这个第一天的值，可以使用SET DATEFIRST命令。例如：

```
SET DATEFIRST 7
```

修改值的时候，7是星期日，1是星期一，依此类推。它会影响DATEPART和DATEADD函数dw（星期几）代码的返回值。

8.2.2 查看当前会话使用的语言

@@LANGID系统函数返回一个表示当前用户会话的本地语言标识符的smallint数据类型值，@@LANGUAGE系统函数返回语言名。

本例返回当前查询会话中目前使用的本地语言设置：

```
SELECT  @@LANGID LanguageID,
      @@LANGUAGE Language
```

这个查询返回：

```
LanguageID      Language
0               us_english
```

在下一个查询中，将使用SET LANGUAGE命令来配置新会话的默认语言，然后再检验语言ID与名称：

```
SET LANGUAGE 'Español'

SELECT  @@LANGID LanguageID,
      @@LANGUAGE Language
```

这个查询返回：

```
Se cambió la configuración de idioma a Español.

LanguageID      Language
5               Español
```

解析

这个技巧演示了返回SQL Server实例的语言。根据用于创建SQL Server实例的区域设置和排序规则的不同，你的默认值也会相同。我也执行了SET LANGUAGE来更改会话的默认语言，它最终影响系统消息的语言以及datetime数据类型数据的格式。

8.2.3 查看和设置当前连接锁超时设置

SET LOCK_TIMEOUT命令配置当前会话中语句等待其他连接释放锁的毫秒数。@@LOCK_TIMEOUT函数用于以毫秒为单位显示当前连接锁超时设置。

本例演示设置和查看当前会话的锁超时值：

```
-- 1000毫秒，1秒
SET LOCK_TIMEOUT 1000
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SELECT @@LOCK_TIMEOUT

-- 没有限制
SET LOCK_TIMEOUT -1
```

这个查询返回：

```
1000
```

解析

本技巧中的示例一开始把锁超时时间设置为1 000毫秒。要使用@@LOCK_TIMEOUT查看修改。之后，又把锁超时时间改为-1来表示无限等待。锁超时时间值告诉我们语句会等待锁定资源多长时间，如果超过了阈值时间，会自动取消语句，然后返回一个错误消息。

8.8.4　显示当前存储过程上下文的嵌套级别

@@NESTLEVEL返回存储过程上下文当前的嵌套级别。存储过程嵌套级别代表某个存储过程调用另外一个存储过程的次数。SQL Server允许存储过程组成最多32次嵌套（非完整）调用。

这个技巧演示如何捕获存储过程上下文的当前嵌套级别（参阅第10章）：

```
-- 第一个过程
CREATE PROCEDURE dbo.usp_QuickAndDirty
AS
SELECT @@NESTLEVEL
GO

-- 第二个过程
CREATE PROCEDURE dbo.usp_Call_QuickAndDirty
AS
SELECT @@NESTLEVEL
EXEC dbo.usp_QuickAndDirty
GO
```

295　创建两个存储过程之后，在调用usp_Call_QuickAndDirty存储过程之前使用@@NESTLEVEL函数：

```
-- 返回0嵌套级别
SELECT @@NESTLEVEL

-- 返回1和2嵌套级别
EXEC dbo.usp_Call_QuickAndDirty
```

这个查询返回3个结果集：

```
0
1
2
```

解析

在这个技巧中，创建了两个存储过程，第一个存储过程usp_QuickAndDirty执行了@@NESTLEVEL。第二个存储过程也调用了@@NESTLEVEL，然后执行了第一个存储过程。在调用过程之前，@@NESTLEVEL返回了0。每次执行嵌套后，@@NESTLEVEL值都增加。

8.8.5　返回当前 SQL Server 实例名和 SQL Server 版本

@@SERVERNAME显示本地服务器名，@@VERSION返回SQL Server实例的版本、日期和处理器信息。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

本例返回当前SQL Server实例名和版本信息：

```
SELECT  @@SERVERNAME ServerName,
      @@VERSION VersionInformation
```

解析

在这个技巧中，演示了返回当前SQL Server实例名和版本信息。和之前的系统配置函数一样，不需要任何参数。

8.8.6 返回当前连接的会话 ID（SPID）

@@SPID返回当前连接的会话ID，可以使用它来识别sp_who系统存储过程或动态管理视图sys.dm_exec_sessions中的额外信息。

这个技巧返回当前SQL连接的服务器进程标识符：

```
SELECT  @@SPID  SPID
```

这个查询返回：

```
SPID
53
```

解析

在这个技巧中，演示了返回当前连接的查询会话的SPID。

296

8.8.7 返回打开事务的数量

@@TRANCOUNT系统函数显示当前连接的活动事务。可以使用这个函数来检测当前会话中打开事务的数量，并且根据这个信息来相应地COMMIT或ROLLBACK事务。这个技巧表明了如何返回当前连接中活动事务的数量：

```
BEGIN TRAN t1
SELECT @@TRANCOUNT -- 返回1
      BEGIN TRAN t2
      SELECT @@TRANCOUNT -- 返回2
         BEGIN TRAN t3
         SELECT @@TRANCOUNT -- 返回3
         COMMIT TRAN
      SELECT @@TRANCOUNT -- 返回2
      ROLLBACK TRAN
SELECT @@TRANCOUNT -- ROLLBACK之后总是返回0!
```

这个查询返回：

```
1
2
3
2
0
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个技巧中，每次发出BEGIN TRAN，@@TRANCOUNT的值都会递增。每次COMMIT TRAN发生的时候，@@TRANCOUNT都会递减。ROLLBACK TRAN执行后，@@TRANCOUNT被设置为0。不管事务嵌套了几层，ROLLBACK TRAN都回滚会话所有打开的事务。

8.8.8　获取之前语句影响的行数

@@ROWCOUNT返回当前范围中上一个T-SQL语句影响的行数的整数值。@@ROWCOUNT_BIG返回bitint值。

在这个示例中，会演示如何返回之前T-SQL语句影响的行：

```
SELECT TOP 3 ScrapReasonID
FROM Production.ScrapReason

297

SELECT @@ROWCOUNT Int_RowCount, ROWCOUNT_BIG() BigInt_RowCount
```

这个查询返回两个结果集：

```
ScrapReasonID
1
2
4

Int_RowCount          BigInt_RowCount
3                     3
```

解析

在这个示例中，第一个语句从Production.ScrapReason表返回了3行，因此@@ROWCOUNT被设为3表示返回了3行。ROWCOUNT_BIG函数和@@ROWCOUNT差不多，只是它可以返回bigint数据类型的总数，而不是@@ROWCOUNT那样的整数数据类型。@@ROWCOUNT和@@ROWCOUNT_BIG通常用于错误处理，例如检查以确保期望数量的行被之前的语句所影响（见第16章）。

8.8.9　获取系统统计信息

SQL Server有几个内建的系统统计函数，表8-9中有描述。

表8-9　系统统计函数

函　数	描　述
@@CONNECTIONS	返回自SQL Server实例上次启动后连接的数量
@@CPU_BUSY	显示自SQL Server实例上次启动后CPU繁忙的毫秒数
@@IDLE	显示自SQL Server实例上次启动后总的空闲时间的毫秒数
@@IO_BUSY	显示自SQL Server实例上次启动后花费在I/O操作上的毫秒数
@@PACKET_ERRORS	显示自SQL Server实例上次启动后总的网络包错误数
@@PACK_RECEIVED	显示自SQL Server实例上次启动后从网络上读取的总的输入包数。你可以监视这个数值是在增加还是保持不变，从而猜测是否有网络问题
@@PACK_SENT	显示自SQL Server实例上次启动后发送到网络上的总的输出包数
@@TIMETICKS	显示每一个tick的毫秒数。*tick*是由特定的毫秒数指定的度量单位（对于Windows系统是31.25毫秒）

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

函　数	描　述
@@TOTAL_ERRORS	显示自SQL Server实例上次启动后遇到的读/写错误
@@TOTAL_READ	显示自SQL Server实例上次启动后非缓存的磁盘读取次数
@@TOTAL_WRITE	显示自SQL Server实例上次启动后的磁盘写入数量

298

本例演示在一个查询中使用系统统计函数：

```
SELECT  'Connections' FunctionNM, @@CONNECTIONS Value
UNION
SELECT  'CPUBusy', @@CPU_BUSY
UNION
SELECT  'IDLE', @@IDLE
UNION
SELECT   'IOBusy', @@IO_BUSY
UNION
SELECT   'PacketErrors', @@PACKET_ERRORS
UNION
SELECT   'PackReceived', @@PACK_RECEIVED
UNION
SELECT   'PackSent', @@PACK_SENT
UNION
SELECT   'TimeTicks', @@TIMETICKS
UNION
SELECT   'TotalErrors', @@TOTAL_ERRORS
UNION
SELECT   'TotalRead', @@TOTAL_READ
UNION
SELECT   'TotalWrite', @@TOTAL_WRITE
```

这个查询返回：

```
FunctionNM      Value
Connections     369
CPUBusy         3333
IDLE            6374793
IOBusy          1916
PacketErrors    2
PackReceived    3606
PackSent        6592
TimeTicks       31250
TotalErrors     0
TotalRead       4688
TotalWrite      5542
```

解析

这个技巧演示了一个SELECT查询引用多个系统统计函数。可以使用它们来跟踪SQL Server实例的各种统计信息。

8.8.10 显示数据库和 SQL Server 设置

使用DATABASEPROPERTYEX系统函数可以获取有关数据库选项的信息。DATABASEPROPERTYEX使用下面的语法：

```
DATABASEPROPERTYEX ( database , property )
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第一个参数是希望探测的数据库名。第二个参数是希望查找的数据库属性。

299 本例演示了如何获取AdventureWorks数据库的排序规则、状态和恢复模式：

```
SELECT DATABASEPROPERTYEX('AdventureWorks', 'Collation'),
       DATABASEPROPERTYEX('AdventureWorks', 'Recovery'),
       DATABASEPROPERTYEX('AdventureWorks', 'Status')
```

这个查询返回：

```
SQL_Latin1_General_CP1_CI_AS SIMPLE ONLINE
```

使用SERVERPROPERTY系统函数可以获取有关SQL Server实例的信息。在它的语法中不需要指定数据库，所以只有一个属性名参数：

```
SERVERPROPERTY (propertyname )
```

这个示例演示返回实例的版本和默认排序规则：

```
SELECT SERVERPROPERTY ('Collation'),
SERVERPROPERTY ('Edition')
```

这个查询返回：

```
SQL_Latin1_General_CP1_CI_AS     Enterprise Edition
```

解析

DATABASEPROPERTYEX和SERVERPROPERTY都能用于获取重要的系统配置信息。在两个实例中，我们都在查询的SELECT子句中引用了函数。

注解　本书中演示了如何使用这些函数，但是没有列举可用属性的列表。对于完整列表，可以参阅SQL Server联机丛书中的SERVERPROPERTY和DATABASEPROPERTYEX主题。

8.8.11　返回当前数据库 ID 和名称

DB_ID函数返回数据库整数ID，DB_NAME返回当前数据库的名字（除非指定了参数）。

本例演示如何获取当前数据库系统ID和名称：

```
SELECT DB_ID() DatabaseID, DB_NAME() DatabaseNM
```

这个查询返回：

```
DatabaseID     DatabaseNM
8              AdventureWorks
```

解析

在这个示例中，内部数据库ID（数据库创建的时候由SQL Server分配）和数据库名一起返回。函

300 数会根据当前数据库上下文返回信息。

它们也都可以接受参数，例如：

```
SELECT DB_ID('master') DatabaseID, DB_NAME(1) DatabaseNM
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DatabaseID    DatabaseNM
1             master
```

使用这些函数的参数可以在不把数据库上下文切换到实际的数据库的情况下查找显式的数据库ID或名称值。

提示 也可以仅查询sys.databases来检索name和database_id。

8.8.12 返回数据库对象名和 ID

OBJECT_ID返回数据库内部指定的数据库对象标识符。OBJECT_NAME根据对象标识符返回对象名。在这个示例中，会演示如何返回数据库对象的名称和ID：

```
SELECT OBJECT_ID('AdventureWorks.HumanResources.Department'),
       OBJECT_NAME(773577794, DB_ID('AdventureWorks'))
```

这个查询返回：

```
757577737 DF_Department_ModifiedDate
```

解析

OBJECT_NAME和OBJECT_ID通常与引用数据库对象标识符的系统目录视图或系统函数一起使用。OBJECT_ID函数用于找出某个对象的内部数据库标识符（需要注意的是，对象ID只在特定数据库中是唯一的）。它的第一个参数是对象的名称。第二个可选参数是对象类型，例如，U表示用户定义表，V表示视图，PK表示主键，还有其他一些值可以在sys.objects目录视图的type列中引用。

OBJECT_NAME用于返回某个对象标识符的对象名。第一个参数是对象ID。第二个可选参数是数据库ID——这在跨数据库的不同对象为同一ID时是有用的。

8.8.13 返回当前用户会话的应用程序和主机

在这个技巧中，会演示用于返回有关当前连接上下文信息的不同函数。APP_NAME返回当前SQL Server连接的应用程序名。HOST_ID返回当前连接的工作站标识号，HOST_NAME返回当前连接的工作站名。

本例演示了如何显示当前连接到SQL Server实例的应用程序名和主机：

```
SELECT APP_NAME() as 'Application',
       HOST_ID() as 'Host ID',
       HOST_NAME() as 'Host Name'
```

这个查询返回：

```
Application                                    Host ID    Host Name
Microsoft SQL Server Management Studio - Query 3388       CAESAR
```

解析

本例的这3个函数都用在SELECT子句中，并且不需要任何参数。如果需要跟踪客户端和应用程序连接的信息，这个信息就很有用，它能帮助你建立标识。

8.8.14 获取当前用户和登录名上下文

SYSTEM_USER函数返回Windows或SQL登录名，USER函数返回当前用户的数据库用户名。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在第一个示例中，会演示如何返回当前用户以及登录名上下文：

```
SELECT  SYSTEM_USER, -- 登录名
      USER -- 数据库用户
```

这个查询返回：

```
CAESAR\Administrator dbo
```

这两个函数还能用作表的DEFAULT值，如下面示例所演示的：

```
CREATE TABLE #TempExample
   (ExampleColumn varchar(10) NOT NULL,
    ModifiedByLogin varchar(55) NOT NULL DEFAULT SYSTEM_USER,
   ModifiedByUser varchar(55) NOT NULL DEFAULT USER)
GO

INSERT #TempExample
(ExampleColumn)
VALUES ('Value A')

SELECT ExampleColumn, ModifiedByLogin, ModifiedByUser
FROM #TempExample
```

这个查询返回下面的结果：

```
ExampleColumn        ModifiedByLogin          ModifiedByUser
Value A              CAESAR\Administrator     dbo
```

302

解析

在这个技巧中，在普通查询中使用了SYSTEM_USER和USER函数，还把它们作为表的DEFAULT值。这些函数对数据库修改审计很有用——例如发生数据修改的时候能捕获当前用户。

8.8.15　查看用户连接选项

在这个技巧中，会演示如何使用SESSIONPROPERTY函数查看当前用户连接的SET属性（更多有关SET选项的信息请参阅第22章）：

```
SELECT SESSIONPROPERTY ('ANSI_NULLS') ANSI_NULLS,
   SESSIONPROPERTY ('ANSI_PADDING') ANSI_PADDING,
   SESSIONPROPERTY ('ANSI_WARNINGS') ANSI_WARNINGS,
   SESSIONPROPERTY ('ARITHABORT') ARITHABORT,
   SESSIONPROPERTY ('CONCAT_NULL_YIELDS_NULL')   CONCAT_NULL_YIELDS_NULL,
   SESSIONPROPERTY ('NUMERIC_ROUNDABORT') NUMERIC_ROUNDABORT,
   SESSIONPROPERTY ('QUOTED_IDENTIFIER') QUOTED_IDENTIFIER
```

这个查询返回如下结果（为了便于阅读，修改了格式）：

```
ANSI_NULLS           ANSI_PADDING              ANSI_WARNINGS
1                    1                         1
 ARITHABORT          CONCAT_NULL_YIELDS_NULL
1                    1
NUMERIC_ROUNDABORT   QUOTED_IDENTIFIER
0                    1
```

解析

使用SESSIONPROPERTY可以查看当前用户的各种数据库连接设置。它接受一个参数，就是需要查看

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的属性名。如果选项是ON，这个函数返回1，如果是OFF则返回0。

8.9 IDENTITY 和 uniqueidentifier 函数

在本章最后的3个技巧中，会介绍如何为表使用IDENTITY值以及如何生成新的uniqueidentifier值。

如果回顾第4章的话就知道，IDENTITY列属性定义在表的某个列上，并且允许我们为表中的一个列定义自增的数字值。

和确保表中唯一的IDENTITY列不同，ROWGUIDCOL属性确保更高级别的唯一。这种唯一的ID保存在uniqueidentifier数据类型中，并且由NEWID系统函数生成。也可以使用NEWSEQUENTIALID系统函数来产生uniqueidentifier返回类型；但是，它与NEWID不同，因为每个新生成的GUID都大于其他服务器范围内生成的GUID。因为NEWSEQUENTIALID在每次执行时都产生更大的值，这种行为会减少键的页分裂和随机页查找。

303

8.9.1 返回最后一个标识值

在这个技巧中，会演示使用3种方法来返回最后生成的标识值。在第一个示例中，使用IDENT_CURRENT函数来返回某个表最后生成的标识值。本命令接受一个参数：要运算的表名。

```
SELECT IDENT_CURRENT('Production.Product') LastIdentityValue
```

这个查询返回：

```
LastIdentityValue
999
```


接着，向定义了IDENTITY列的表中新插入一行。在INSERT后立即使用SCOPE_IDENTITY和@@IDENTITY函数（两个函数的区别在示例后描述）获取最后生成的标识值：

```
-- 示例插入，在表中生成IDENTITY值
INSERT HumanResources.Department
(Name, GroupName)
VALUES ('TestDept', 'TestGroup')

-- 任何表生成的最后标识值
-- 在当前会话中，当前作用域中
SELECT SCOPE_IDENTITY()
```

该例返回当前会话当前作用域中表INSERT生成的最后标识值。作用域代表如果INSERT引起触发器触发并且插入另外一行到另一个基于IDENTITY的表中，我们只会看到当前会话中（不是作用域之外的触发器会话）最后的IDENTITY值：

```
17
```

执行@@IDENTITY会产生当前会话任何作用域中的任何表生成的IDENTITY值：

```
-- 任何表生成的最后标识值
-- 在当前会话中，任何域中
SELECT @@IDENTITY
```

这个查询返回：

```
17
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

尽管对于本示例的查询来说值是一样的，但是如果INSERT触发了触发器并引起另外一个基于IDENTITY的表的INSERT操作，那么我们会看到触发器作用域中另外一个表最后的标识值。

解析

本技巧演示了3种返回最后生成的标识值的方法。第一个查询使用了IDENT_CURRENT，它指定某个表最后生成的标识值。

演示的第二个函数SCOPE_IDENTITY针对当前用户的会话，并且返回当前作用域中最后生成的值。
304 例如，当前作用域指的是当前SQL语句、当前存储过程或者当前触发器。

而@@IDENTITY返回当前会话中任何表最后生成的值，它是跨所有作用域的。所以，如果当前作用域中的INSERT触发了一个触发器，而它又插入一个记录到其他表，则@@IDENTITY会返回受到触发器影响的插入行最新的值，而不是你期望捕获的原来插入的值。

简而言之，如果希望获取跨任何会话或作用域的某个表的最新IDENTITY值，可以使用IDENT_CURRENT。如果你希望获取当前作用域和当前会话中的任何表的最新IDENTITY值，可以使用SCOPE_IDENTITY。如果希望得到当前会话任何作用域中任何表的最后IDENTITY值，可以使用@@IDENTITY。

8.9.2　返回标识列的种子和增量值

IDENT_INCT函数显示某个表或者引用视图的IDENTITY列的原始增量值。IDENT_SEED显示某个表或引用视图的IDENTITY列上原始定义的种子值。这些函数对于检测插入行时IDENTITY列的增量和种子值很有用。

本例演示了返回某个表的标识增量和种子：

```
SELECT IDENT_INCR('Production.Product') IdentIncr,
       IDENT_SEED('Production.Product') IdentSeed
```

这个查询返回：

```
IdentIncr     IdentSeed
1             1
```

解析

在这个技巧中，我们使用IDENT_INCR和IDENT_SEED返回Production.Product表的增量和种子值。

8.9.3　创建新的 uniqueidentifier 值

NEWID函数用于创建uniqueidentifier数据类型的值。第一个示例在SELECT语句中返回一个新的uniqueidentifier值：

```
SELECT NEWID()
```

这个查询返回一个类似下面的值（你的结果会与这里的不同）：

```
D04ED24F-671E-4559-A205-F6864B9C59A7
```

然后创建一个使用NEWID函数作为默认值的新临时表：

```
CREATE TABLE #T4
   (MyValue uniqueidentifier NOT NULL DEFAULT NEWID())
```

305 然后，插入一个新值到表中：

```
INSERT #T4 DEFAULT VALUES
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

最后，从表中获取这个值：

```
SELECT MyValue
FROM #T4
```

这个查询返回：

```
MyValue
2DD54CE0-5D26-42F9-A68D-7392DB89D0EF
```

NEWSEQUENTIALID也可以用来生成新的GUID值，只有在这种情况下，计算机上每个新生成的值才会都大于之前生成的任何值。

为了演示它，首先会创建新的临时表，并且插入5行：

```
CREATE TABLE #T5
   (MyValue uniqueidentifier NOT NULL DEFAULT NEWSEQUENTIALID(),
    InsertDT datetime2 NOT NULL DEFAULT SYSDATETIME())
GO

INSERT #T5 DEFAULT VALUES
INSERT #T5 DEFAULT VALUES
INSERT #T5 DEFAULT VALUES
INSERT #T5 DEFAULT VALUES
INSERT #T5 DEFAULT VALUES
```

接着，我将查询表，并以uniqueidentifier值进行排序：

```
SELECT MyValue, InsertDT
FROM #T5
ORDER BY MyValue
```

注意值的顺序也与插入时日期的顺序相匹配：

```
MyValue                              InsertDT
EE78AB60-E548-DC11-A195-00188B28C9C5 2007-08-12 10:04:53.0833262
EF78AB60-E548-DC11-A195-00188B28C9C5 2007-08-12 10:04:53.8033262
F078AB60-E548-DC11-A195-00188B28C9C5 2007-08-12 10:04:54.3603262
F178AB60-E548-DC11-A195-00188B28C9C5 2007-08-12 10:04:54.9103262
F278AB60-E548-DC11-A195-00188B28C9C5 2007-08-12 10:04:55.3903262
```

解析

如这个技巧所展示的，NEWID和NEWSEQUENTIALID可以用在SELECT语句中，也可以在CREATE或ALTER TABLE语句中作为DEFAULT列值。但NEWID提供了随机值，而NEWSEQUENTIALID提供增长的uniqueidentifier值。这个行为减少了键上的页分裂和随机页查找。

警告 与NEWID不同，NEWSEQUENTIALID产生的增长的值基于存在的值。例如，如果GUID暴露在URL中，一个人想看其他人的数据，他就可以通过尝试增加自己的值来查看连续的记录。

还要注意的是，如果SQL Server实例没有网卡（这不是通常的配置），会在接触的服务器作用域中生成唯一值——意思是在其他SQL实例中会生成重复的值。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第9章 条件处理、流控制和游标

在本章中，会介绍一些技巧来演示用于下列操作的SQL Server T-SQL。

- 条件处理（conditional processing）。你将了解如何使用CASE和IF...ELSE语句来计算条件并返回相应的值。我会介绍如何使用CASE函数来计算单个输入表达式并返回一个值，以及如何计算一个或多个布尔表达式。最后，会演示当表达式为TRUE的时候返回一个值。
- 流控制功能（control-of-flow functionality）。这个技巧演示如何根据诸如RETURN、WHILE、WAITFOR和GOTO之类的命令控制T-SQL语句或批处理的执行。RETURN用来立即退出当前的T-SQL批处理，禁止之后的任何代码执行。WHILE命令用于在条件为TRUE的时候重复某个操作或者一批操作。WAITFOR命令用于为T-SQL代码的执行延迟一段时间或延至某个时间点。GOTO用于跳转到T-SQL批处理中的某个标签，忽略GOTO之后的代码。
- 创建和使用游标（creating and using cursor）。在这里会演示T-SQL游标，可用来一次处理一行。基于我在这个领域的经验，我不是游标的全力支持者。由于过度的单行调用和内存消耗，或没有正确处理带来的代码量问题，游标就会引发重大性能问题。然而，还有一些少量的场合使用游标会比基于集合的方案更好。

理解如何以及何时（何时不）使用这些技术能让你创建灵活且智能的T-SQL代码。

9.1 条件处理

条件处理可以根据一个或者一组表达式的值来返回结果。下面的几个技巧会演示SQL Server的条件处理命令，包括CASE和IF...ELSE。

CASE函数用于根据一个表达式的值返回结果。它常用于将代码转换为描述性的值或者计算多个条件来返回一个值（例如，“如果行是2008年的并且小于或等于当前的季度，则返回销售总数”）。

IF...ELSE结构计算一个布尔表达式，如果是TRUE则执行一个T-SQL语句或批处理。这个命令有很多用途，使用它可以根据条件返回结果集、更新数据或根据一个或多个搜索条件执行存储过程。

下面3个技巧会实战演示条件处理。

9.1.1 使用 CASE 来计算单个输入表达式

CASE函数用于根据一个表达式的值返回结果。它同样能用于根据一个或多个布尔表达式的结果返回值。前一种CASE的语法如下：

```
CASE input_expression
    WHEN when_expression THEN result_expression
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    [ ...n ]
     [
    ELSE else_result_expression
     ]
END
```

表9-1描述了此命令的参数。

表9-1 CASE的参数

参 数	描 述
input_expression	要在CASE语句中计算的输入值
when_expression	要和input_expression比较的表达式。例如，如果input_expression是Gender列，when_expression就可能是'F'或'M'。如果input_expression和when_expression之间存在匹配，就返回result_expression
result_expression	input_expression和when_expression相等时要返回的值

这个示例演示如何使用CASE来计算一个或多个条件，根据计算为TRUE的那些条件返回结果：

```
USE AdventureWorks
GO

-- 根据部门决定会议室
SELECT DepartmentID,
       Name,
       GroupName,
       CASE GroupName
          WHEN 'Research and Development' THEN 'Room A'
          WHEN 'Sales and Marketing' THEN 'Room B'
          WHEN 'Manufacturing' THEN 'Room C'
          ELSE 'Room D'
       END ConferenceRoom
FROM HumanResources.Department
```

这个查询返回下面的（部分）结果：

```
DepartmentID    Name            GroupName                   ConferenceRoom
1               Engineering     Research and Development    Room A
2               Tool Design     Research and Development    Room A
3               Sales           Sales and Marketing         Room B
4               Marketing       Sales and Marketing         Room B
5               Purchasing      Inventory Management        Room D
...
```

解析

在这个技巧的示例中，CASE用来根据GroupName值分配一个会议室。CASE语句跟在SELECT子句中的Name列后面：

```
SELECT DepartmentID,
       Name,
       GroupName,
      CASE GroupName
```

要计算的列GroupName跟在CASE关键字后面。接着，计算一组WHEN表达式。每一个部门根据GroupName值分配了不同的房间：

```
WHEN 'Research and Development' THEN 'Room A'
WHEN 'Sales and Marketing' THEN 'Room B'
WHEN 'Manufacturing' THEN 'Room C'
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

没有一个WHEN表达式计算为TRUE，则可选的ELSE子句用来指定一个默认的结果表达式：

```
ELSE 'Room D'
```

END关键字用于标记CASE语句的结束，在这个技巧中，它后面跟随列别名：

```
END ConferenceRoom
```

9.1.2 使用 CASE 来计算布尔表达式

CASE提供了另外一种语法，它不使用初始的输入表达式，而是使用一个或多个布尔表达式，当表达式计算为TRUE的时候返回一个结果表达式。语法如下：

```
CASE
     WHEN Boolean_expression THEN result_expression
    [ ...n ]
     [
    ELSE else_result_expression
     ]
END
```

和前面的技巧相比，语法中另外的参数是要计算的表达式boolean_expression。每一个WHEN都计算一个布尔表达式而不是输入表达式，如果TRUE则返回结果表达式。这种形式的CASE允许额外的表达式，而不是仅仅计算一个输入表达式的值。

如果没有表达式计算为TRUE，则返回ELSE子句的result_expression，或者如果没有指定ELSE子句的话就返回NULL值。如果一行与多个布尔表达式匹配，那么第一个计算为TRUE的布尔表达式决定了结果表达式。在这个示例中，除了计算部门名外，还要计算其他表达式，例如部门标识号和以字母T开头的房间名：

309

```
SELECT DepartmentID,
      Name,
      CASE
         WHEN Name = 'Research and Development'
            THEN 'Room A'
         WHEN (Name = 'Sales and Marketing' OR
               DepartmentID = 10)
            THEN 'Room B'
         WHEN Name LIKE 'T%'
            THEN 'Room C'
         ELSE 'Room D'
      END ConferenceRoom
FROM HumanResources.Department
```

这个查询返回如下的（部分）结果：

DepartmentID	Name	ConferenceRoom
12	Document Control	Room D
1	Engineering	Room D
16	Executive	Room D
14	Facilities and Maintenance	Room D
10	Finance	Room B
9	Human Resources	Room D
...		
6	Research and Development	Room A
3	Sales	Room D
15	Shipping and Receiving	Room D
17	TestDept	Room C
2	Tool Design	Room C

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个示例中，使用了3个布尔表达式。如果部门名是Research and Development，则返回Room A：

```
WHEN Name = 'Research and Development'
          THEN 'Room A'
```

第二个布尔表达式规定，如果部门名是Sales and Marketing或者DepartmentID等于10，则返回Room B：

```
WHEN (Name = 'Sales and Marketing' OR
            DepartmentID = 10)
          THEN 'Room B'
```

第三个布尔表达式查找所有以字母T开头的部门名，如果存在匹配则返回Room C：

```
WHEN Name LIKE 'T%'
     THEN 'Room C'
```

9.1.3 使用 IF...ELSE

IF...ELSE计算一个布尔表达式，如果是TRUE则执行一个T-SQL语句或批处理。

语法如下：

310

```
IF Boolean_expression
    { sql_statement | statement_block }
[ ELSE
    { sql_statement | statement_block } ]
```

如果布尔表达式计算为FALSE，则调用ELSE子句，执行ELSE后的T-SQL语句或批处理。

这个示例演示了根据局部变量的值有条件地执行查询：

```
DECLARE @QuerySelector int
SET @QuerySelector = 3

IF @QuerySelector = 1
BEGIN
   SELECT TOP 3
       ProductID, Name, Color
   FROM Production.Product
   WHERE Color = 'Silver'
   ORDER BY Name
END
ELSE
BEGIN
   SELECT TOP 3
      ProductID, Name, Color
   FROM Production.Product
   WHERE Color = 'Black'
   ORDER BY Name
END
```

这个查询返回：

```
ProductID     Name                    Color
322           Chainring               Black
863           Full-Finger Gloves, L   Black
862           Full-Finger Gloves, M   Black
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个技巧中，创建了一个叫@QuerySelector的整型局部变量，它被设置为值3：

```
DECLARE @QuerySelector int
SET @QuerySelector = 3
```

IF语句从计算@QuerySelector是否等于1开始：

```
IF @QuerySelector = 1
```

311 如果计算得出@QuerySelector确实是1，则执行下面那段代码（从BEGIN语句开始）：

```
BEGIN
   SELECT TOP 3
       ProductID, Name, Color
   FROM Production.Product
   WHERE Color = 'Silver'
   ORDER BY Name
END
```

如果IF后是单条语句，则BEGIN是可选的。但如果是必须成组执行的多条语句，则必须使用BEGIN和END。作为最佳实践，对于单条语句也最好使用BEGIN和END，这样如果以后代码有修改你就不会忘记这样做。

如果前面的IF条件计算后为FALSE，则可选的ELSE子句搜索黑色的产品：

```
ELSE
BEGIN
   SELECT TOP 3
      ProductID, Name, Color
   FROM Production.Product
   WHERE Color = 'Black'
   ORDER BY Name
END
```

因为@QuerySelector变量为3，所以执行了第二个T-SQL代码块，返回Color = 'Black'的产品。

9.2 流控制

在接下来的几个技巧中，会演示如何使用下列SQL Server 2005流控制函数和命令。

- RETURN：这个函数用于无条件退出既有域并将控制权返回给调用域。RETURN也能用于把整型值返回给调用者。这个技术通常用于把业务逻辑错误返回给调用的过程，或者确认批处理/查询/域中执行的任何东西都没有发生错误。
- WHILE：当布尔条件计算为TRUE时，可以使用它来重复执行同样的批处理或T-SQL代码。WHILE通常作为游标（本章同样会介绍）的替代方式，因为可以使用它遍历结果集，一次为一行产生行为，直到结果集变空为止。例如，可以把一些碎片程度大于50%的索引名填充到一个临时表。调用WHILE语句能保证只要表中有行就进行循环。对于每一次迭代，可以得到第一个索引行，并在从表中捕获的第一个索引名上进行索引重建。之后，就可以从表中删除那行，并继续遍历索引，直到表变空才结束WHILE循环。
- GOTO：这个函数能用于在T-SQL批处理中跳转到一个标签。它通常用于在发生错误的时候跳转到某个错误处理器，或者跳过某个满足或不满足条件的代码。GOTO早就赢得了“面条式”代码的坏名声。这是因为你为了完全理解这段代码或过程实际在做什么，不得不在代码块间跳来跳去。尽管应该尽量少用GOTO，但GOTO还是被支持的，因此我们也会在技巧中介绍。

312

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- WAITFOR：可以使用这个函数来延迟它后面的后续T-SQL命令的处理——可以是一个固定的时间段或是到某个时间。这在活动是同步的情况下非常有用。例如，如果代码只有当外部任务结束（需要几秒/几分钟/几小时）后才能完成，或者如果你希望在指定时间进行某个活动（例如，非营业时间）。

9.2.1 使用 RETURN

RETURN用来立即退出当前的T-SQL批处理、查询或存储过程，并且不执行它之后的批处理/查询/过程域中的任何代码。RETURN仅退出当前域中执行的代码。如果在存储过程A中调用了存储过程B，并且存储过程B发起一个RETURN，那么存储过程B会立即终止，但是存储过程A还会继续，就好像B已经成功完成。

本例演示了如何使用RETURN来无条件终止一个查询：

```
IF NOT EXISTS
(SELECT ProductID FROM Production.Product WHERE Color = 'Pink')
BEGIN
   RETURN
END

-- 不会再执行
SELECT ProductID
FROM Production.Product
WHERE Color = 'Pink'
```

这个查询返回：

```
命令已成功完成。
```

RETURN同样支持可选的整数表达式：

```
RETURN [ integer_expression ]
```

这个整数值能用于存储过程和调用应用程序的通信，例如：

```
-- 创建一个产生逻辑错误的存储过程
CREATE PROCEDURE #usp_TempProc
AS
SELECT 1/0
RETURN @@ERROR
GO
```

接着，执行存储过程，从局部变量捕获RETURN代码：

```
DECLARE @ErrorCode int

EXEC @ErrorCode = #usp_TempProc
PRINT @ErrorCode
```

这个查询返回被零除的错误，错误后面显示错误号码：

```
消息8134，级别16，状态1，过程
#usp_TempProc_____00000B72，第4行
遇到以零作除数错误。
8134
```

解析

在这个技巧中，IF条件检测是否存在粉红色的产品：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
IF NOT EXISTS
(SELECT ProductID FROM Production.Product WHERE Color = 'Pink')
```

如果计算得到TRUE（不存在粉红色的产品），则执行RETURN语句：

```
BEGIN
   RETURN
END

-- 不会再执行
SELECT ProductID
FROM Production.Product
WHERE Color = 'Pink'
```

因为没有粉红色的产品，所以调用了RETURN，且跟在IF语句后的SELECT查询永远不会被执行。

第二个示例演示了创建一个临时存储过程，它包含了创建以0作除数的错误的T-SQL。RETURN用来捕获@@ERRORCODE值8134，它返回给调用者并通过@ErrorCode局部变量打印出来。如果整数值没有显式放入RETURN的调用，那么会发送0作为默认值。

9.2.2　使用 WHILE

在这个技巧中，会演示WHILE命令，只要条件是TRUE，它就能重复某个操作或一批操作。

WHILE的语法如下：

```
WHILE Boolean_expression
     { sql_statement | statement_block }
      [ BREAK ]
     { sql_statement | statement_block }
      [ CONTINUE ]
     { sql_statement | statement_block }
```

只要布尔表达式还是TRUE，WHILE就会保证T-SQL语句或者批处理一直处理。使用BREAK关键字可以从最内部的WHILE循环中退出，CONTINUE关键字导致循环重启动。

在这个示例中，系统存储过程sp_spaceused用来返回@AWTables表变量中每个表的空间占用情况：

```
-- 声明变量
DECLARE @AWTables TABLE (SchemaTable varchar(100))
DECLARE @TableName varchar(100)

-- 把表名插入表变量
INSERT @AWTables
(SchemaTable)
SELECT TABLE_SCHEMA + '.' + TABLE_NAME
FROM INFORMATION_SCHEMA.tables
WHERE TABLE_TYPE = 'BASE TABLE'
ORDER BY TABLE_SCHEMA + '.' + TABLE_NAME

-- 使用sp_spaceused汇报每个表的空间使用情况
WHILE (SELECT COUNT(*) FROM @AWTables)>0
BEGIN
   SELECT TOP 1 @TableName = SchemaTable
   FROM @AWTables
   ORDER BY SchemaTable

   EXEC sp_spaceused @TableName

   DELETE @AWTables
   WHERE SchemaTable = @TableName
```

314

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
END
```

这个查询返回多个结果集（一个结果集对应一个表）。这里显示了3个结果集：

name	rows	reserved	data	index_size	unused
Shift	3	48 KB	8 KB	40 KB	0 KB

name	rows	reserved	data	index_size	unused
Department	20	32 KB	8 KB	24 KB	0 KB

name	rows	eserved	data	index_size	unused
EmployeeAddress	290	48 KB	16 KB	32 KB	0 KB

在前面对WHILE命令的介绍中我们看到，还可以在代码中使用关键字BREAK和CONTINUE。BREAK用来退出WHILE循环，而CONTINUE用来继续WHILE循环。例如：

```
WHILE (1=1)
BEGIN
   PRINT 'Endless While, because 1 always equals 1'
   IF 1=1
   BEGIN
      PRINT 'But we didn''t let the endless loop happen'
      BREAK
   END
   ELSE
   BEGIN
      CONTINUE
   END
END
```

这个查询返回：

```
Endless While, because 1 always equals 1
But we didn't let the endless loop happen
```

解析

在这个技巧中，WHILE用来遍历AdventureWorks数据库中的每一个表，使用sp_spaceused系统存储过程来汇报信息。

这个技巧从声明两个变量开始：

```
DECLARE @AWTables TABLE (SchemaTable varchar(100))
DECLARE @TableName varchar(100)
```

表变量@AWTables用来保存所有的表名，@TableName变量用于保存单个表的名字。

表变量被填充AdventureWorks数据库中的所有表名（填充schema.table_name值）：

```
INSERT @AWTables
(SchemaTable)
SELECT TABLE_SCHEMA + '.' + TABLE_NAME
FROM INFORMATION_SCHEMA.tables
WHERE TABLE_TYPE = 'BASE TABLE'
ORDER BY TABLE_SCHEMA + '.' + TABLE_NAME
```

然后开始了WHILE循环，只要@AWTables表变量中还有行就进行循环：

```
WHILE (SELECT COUNT(*) FROM @AWTables)>0
BEGIN
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在WHILE中，使用TOP1表变量中的第一个表名对@TableName局部变量进行赋值：

```
SELECT TOP 1 @TableName = SchemaTable
FROM @AWTables
ORDER BY SchemaTable
```

使用@TableName变量来执行EXEC sp_spaceused：

```
EXEC sp_spaceused @TableName
```

最后，已经汇报的表的那行从表变量中删除：

```
   DELETE @AWTables
   WHERE SchemaTable = @TableName
END
```

WHILE会继续执行sp_spaceused，直到@AWTables表变量中所有行都被删除。

在技巧的第二个示例中，如果满足了某个条件（或者达到了阈值），我们使用BREAK来退出循环。使用BREAK可以防止无限循环。

9.2.3　使用 GOTO

这个技巧演示了GOTO，它用来跳转到T-SQL批处理中的一个标签，而忽略它之后的代码。语法如下：

```
     GOTO label
label definition: code
```

在这个示例中我们检查部门名是否已被既有的部门使用，如果是则使用GOTO跳过INSERT。如果不是则执行INSERT：

316

```
DECLARE @Name nvarchar(50) = 'Engineering'
DECLARE @GroupName nvarchar(50) = 'Research and Development'
DECLARE @Exists bit = 0

IF EXISTS (SELECT Name
         FROM HumanResources.Department
          WHERE Name = @Name)
BEGIN
   SET @Exists = 1
   GOTO SkipInsert
END

INSERT HumanResources.Department
(Name, GroupName)
VALUES(@Name , @GroupName)

SkipInsert:
IF @Exists = 1
BEGIN
     PRINT @Name + ' already exists in HumanResources.Department'
END
ELSE
BEGIN
     PRINT 'Row added'
END
```

这个查询返回：

```
Engineering already exists in HumanResources.Department
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个技巧中，声明了两个局部变量，并对其设置了值，准备插入到HumanResources.Department表中。

```
DECLARE @Name nvarchar(50) = 'Engineering'
DECLARE @GroupName nvarchar(50) = 'Research and Development'
```

定义另一个变量来保存位值。这个值作为一个标记，标识出行是否已经存在于表中（在技巧后面的内容中使用）：

```
DECLARE @Exists bit = 0
```

接着，使用IF语句来检查是否存在和局部变量有相同部门名的行。如果存在这样的行就调用GOTO命令。GOTO引用了你希望跳转到的标签名，在这里是SkipInsert：

```
IF EXISTS (SELECT Name
        FROM HumanResources.Department
         WHERE Name = @Name)
BEGIN
   SET @Exists = 1
   GOTO SkipInsert
END
```

317

IF语句后是INSERT，然而在示例中它被跳过了，因为HumanResources.Department表中已经存在Engineering部门了：

```
INSERT HumanResources.Department
(Name, GroupName)
VALUES(@Name , @GroupName)
```

然后定义了要跳转到的标签，以一个冒号作为后缀：

```
SkipInsert:
```

在标签后面是另一个IF语句。如果位标记已激活，PRINT语句指定行已经存在。

```
IF @Exists = 1
BEGIN
     PRINT @Name + ' already exists in HumanResources.Department'
END
```

否则，打印一条成功插入行的消息：

```
ELSE
BEGIN
    PRINT 'Row added'
END
```

作为最佳实践，如果可以选择使用GOTO或其他流控制方法的话，应该选择其他的而不是GOTO。GOTO会减少代码的清晰度，因为你不得不在批处理或存储过程中转来转去以理解查询作者原来的意图。

9.2.4 使用 WAITFOR

在这个技巧中，演示了WAITFOR命令，它将T-SQL的执行延迟指定的一段时间：

```
WAITFOR
{
    DELAY 'time_to_pass'
  | TIME 'time_to_execute'
  | ( receive_statement ) [ , TIMEOUT timeout ]
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

WAITFOR DELAY的time_to_pass参数是在执行命令前等待的秒数、分钟数和小时数。WAITFOR TIME的time_to_execute参数用来指定执行批处理的实际时间（小时、分钟和秒）。receive_statement和TIMEOUT选项用来和Service Broker（参见第20章）一起使用。

在第一个示例中，在SELECT查询执行之前通过WAITFOR创建一个10秒的延迟：

```
WAITFOR DELAY '00:00:10'
BEGIN
   SELECT TransactionID, Quantity
   FROM Production.TransactionHistory
END
```

318

在第二个示例中，查询直到指定时间才会执行，在这里是7:01PM：

```
WAITFOR TIME '19:01:00'
BEGIN
   SELECT COUNT(*)
   FROM Production.TransactionHistory
END
```

解析

在这个技巧中，我们使用两个版本的WAITFOR来延迟T-SQL批处理的处理。

第一个查询在执行批处理前等待了10秒：

```
WAITFOR DELAY '00:00:10'
```

如果你知道另外一个操作必须同步执行而当前的批处理必须等待的话，那么等待一段时间是很有用的。例如，如果你使用sp_start_job系统存储过程发出一个同步SQL Server代理作业的话，控制权在作业开始执行后马上返回给批处理。如果你知道刚刚发出的作业至少需要运行5分钟，而之后的任务依赖这个作业的完成，那么就能使用WAITFOR来延迟处理，直到作业完成。

第二个查询一直等到指定时间的下个示例：

```
WAITFOR TIME '19:01:00'
```

当某个操作必须在一天的某个时间段发生的话，则WAIRFOR TIME很有用。例如，假设你有一个存储过程，它为事务处理表进行数据仓库聚合。聚合可能需要几个小时来完成，但你不希望在营业时间之后才从生产表中加载完成的数据。在过程中使用WAITFOR TIME就可以在非营业时间之前停止表的最终加载。

9.3　游标

有编程背景的查询编写者通常更习惯于使用游标，而不是基于集合的方案来获取或更新行。例如，程序员可能会决定一次遍历一行，一条一条地更新行，而不是用一个操作来更新整个行集。但是，游标会耗尽SQL Server实例的内存、减少并发性、减少网络带宽、锁定资源，并且经常会需要比基于集合的方案更多的代码。T-SQL是基于集合的语言，也就是说它擅长操作和获取行集，而不是一行一行地处理。

然而，应用程序或业务需求可能需要T-SQL游标提供的一行一行的处理方式。通常，应该在其他诸如WHILE循环、子查询、临时表或表变量等其他行级别的处理方法都不管用的时候才应该考虑使用游标。

T-SQL游标的大致生存周期如下所示。

- 游标通过返回有效结果集的SQL语句来定义。
- 然后游标被填充（打开）。
- 一旦打开，行就能从游标中获取，一次一个或一组。行还能向前或向后移动，这取决于原始

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

游标的定义。

- 根据游标类型，数据能随着行的滚动进行修改，或读取并用在其他操作中。
- 最后，游标使用后应该显式关闭并从内存中释放。

DECLARE CURSOR命令用来创建游标，而且有很多会影响游标的灵活性和锁定行为的选项。语法如下：

```
DECLARE cursor_name CURSOR
[ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
[ TYPE_WARNING ]
FOR select_statement[ FOR UPDATE [ OF column_name [ ,...n ] ] ]
```

有一些选项会影响游标数据是否能更新以及你是否可以向后向前移动游标中填充的行。表9-2简要描述了可用的一些选项。

表9-2 游标选项

参 数	描 述
LOCAL或GLOBAL	如果选择了LOCAL，游标只在SQL批处理、触发器或存储过程的范围中可用。如果选择了GLOBAL，游标对连接自身可用（例如，一个执行创建游标的存储过程的连接能使用通过执行存储过程创建的游标）
FORWARD_ONLY或SCROLL	FORWARD_ONLY选项只允许我们从游标第一行开始向前移动。而SCROLL允许我们使用所有的FETCH选项（FIRST、LAST、NEXT、PRIOR、ABSOLUTE和RELATIVE）在结果集中前后移动。如果考虑性能，最好使用FORWARD_ONLY——因为这种游标类型的负载比SCROLL低
STATIC或KEYSET或DYNAMIC或FAST_FORWARD	如果指定了STATIC，则游标数据的快照会保存在DYNAMIC或FAST_FORWARD tempdb数据库中，任何对原始数据源的修改不会影响游标数据。KEYSET允许我们看到游标外部对行的修改，但是看不到符合游标SELECT查询的插入或游标打开之后的删除。DYNAMIC允许我们在游标打开时看到基础数据源的更新、插入和删除。FAST_FORWARD定义两种行为：设置游标为只读和只向前状态（通常这是性能最好的游标选项，但是也最不灵活）。如果面临性能问题，并且你的功能不复杂，则使用这个选项
READ_ONLY或SCROLL_LOCKS或OPTIMISTIC	READ_ONLY选项意味着不能通过游标进行更新。如果考虑性能和并发，则使用这个选项。SCROLL_LOCKS在行上加锁，这样就能保证在游标结束后进行更新和删除。OPTIMISTIC选项在已更新或已删除的行上不加锁，只有当上次数据读取之后游标外部没有进行数据更新时才会使修改有效
TYPE_WARNINGS	如果指定了TYPE_WARNINGS，且游标显式地从一种类型转换成了另一种类型，那么会有警告被发送到客户端

select_statement参数是用来在游标中定义数据的查询。避免使用查询来返回很多用不到的列和行，因为游标打开的时候会放在内存中。UPDATE [OF column_name [,...n]]用来指定那些允许游标更新的列。

一旦游标使用DECLARE CURSOR来声明，下一步就是使用OPEN命令打开它并填充数据。语法如下：

```
OPEN { [ GLOBAL ] cursor_name }
```

游标能在局部（默认）或全局打开。一旦打开了，你就可以使用FETCH命令在游标中切换行。FETCH NEXT的语法如下：

```
FETCH [ [ NEXT | PRIOR | FIRST | LAST
                | ABSOLUTE { n | @nvar }
                | RELATIVE { n | @nvar } ]
          FROM ]
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
{ [ GLOBAL ] cursor_name }
[ INTO @variable_name [ ,...n ] ]
```

FETCH提供了几个选项用来在游标中切换行，通过把每个游标定义的列的结果填充到局部变量来实现（在下个技巧会演示）。

@@FETCH_STATUS函数用在FETCH操作后，用来检测FETCH状态，返回0表示成功，-1表示不成功，-2表示丢失。

一旦用完了打开的游标，就执行CLOSE命令从内存中释放结果集。语法如下：

```
CLOSE { [ GLOBAL ] cursor_name }
```

现在，仍然可以重新打开游标。然而，如果已经用完了，应该使用DEALLOCATE命令移除内部系统对游标的引用。这样会释放游标使用的所有资源。例如，如果游标的卷锁被表引用，这些锁在DEALLOCATE后释放。语法如下：

```
DEALLOCATE { [ GLOBAL ] cursor_name }
```

下个技巧会演示每个命令。

创建和使用 T-SQL 游标

尽管本书推荐少使用游标，但是像下个示例演示的那样使用游标定期收集数据库管理信息是很恰当的。

这个技巧演示了使用游标来遍历当前SQL Server实例中活动的每个会话ID，并且执行DBCC OUTPUTBUFFER来查看每一个会话的ASCII和十六进制输出缓冲区（如果当前在执行一些东西的话）：

321

```
-- 不在结果集中显示行数
SET NOCOUNT ON

DECLARE @session_id smallint

-- 声明游标
DECLARE session_cursor CURSOR
FORWARD_ONLY READ_ONLY
FOR SELECT session_id
   FROM sys.dm_exec_requests
   WHERE status IN ('runnable', 'sleeping', 'running')

-- 打开游标
OPEN session_cursor

-- 从游标中一次检索一行
FETCH NEXT
FROM session_cursor
INTO @session_id

-- 当存在游标就持续检索行
WHILE @@FETCH_STATUS = 0
BEGIN

   PRINT 'Spid #: ' + STR(@session_id)
   EXEC ('DBCC OUTPUTBUFFER (' + @session_id + ')')

   -- 提取下一行
   FETCH NEXT
   FROM session_cursor
   INTO @session_id
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
END

-- 关闭游标
CLOSE session_cursor

-- 释放游标
DEALLOCATE session_cursor
```

这个查询返回SQL Server实例的每个活动请求的输出缓冲区。

解析

这个技巧从设置SET NOCOUNT ON开始，它禁止SQL Server显示行数消息来提供更清楚的输出：

```
-- 不在结果中显示行数
SET NOCOUNT ON
```

接着，定义局部变量来保存要从游标中获取的服务器进程ID的值：

```
DECLARE @session_id smallint
```

游标通过使用DECLARE CURSOR来定义。游标包含sys.dm_exec_requests动态管理视图的session_id列：

322

```
-- 声明游标
DECLARE session_cursor CURSOR
FORWARD_ONLY READ_ONLY
FOR SELECT session_id
   FROM sys.dm_exec_requests
   WHERE status IN ('runnable', 'sleeping', 'running')
```

在游标定义后，把它打开（填充）：

```
OPEN session_ cursor
```

一旦打开，使用FETCH NEXT把第一行的值获取到@SPID局部变量中：

```
FETCH NEXT
FROM session_cursor
INTO @session_id
```

FETCH NEXT用于获取第一行。在第一次获取之后，我们定义了WHILE条件来让SQL Server继续循环语句，一直到游标的获取状态不再成功为止（也就是说没有行能获取了）：

```
WHILE @@FETCH_STATUS = 0
BEGIN
```

@@FETCH_STATUS用来返回游标FETCH语句最后发出的打开游标的状态，返回0表示最后一次FETCH成功，-1表示未成功，-2表示丢失。

在WHILE语句中，打印了@session_id变量，并使用EXEC来创建一个动态查询：

```
PRINT 'Spid #: ' + STR(@session_id)
EXEC ('DBCC OUTPUTBUFFER (' + @session_id + ')')
```

动态查询为每一个session_id执行DBCC OUTPUTBUFFER。在这之后，运行另一个FETCH NEXT来填充下一个@session_id值：

```
-- 提取下一行
FETCH NEXT
FROM session_cursor
INTO @session_id
END
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在所有session_id都获取后，WHILE循环就退出了（因为@@FETCH_STATUS会返回-1）。然后使用CLOSE命令来关闭游标：

```
-- 关闭游标
CLOSE session_cursor
```

至此，游标仍然可以使用OPEN命令来打开。要彻底从内存中移除游标，需要使用DEALLOCATE：

```
-- 释放游标
DEALLOCATE session_cursor
```

尽管有用，游标还是应该小心处理，因为它们会消耗大量资源，并且执行效果不如等价的基于集合的方案。在考虑在T-SQL开发中使用游标之前，确保先研究一下所有的基于集合的方案。

323

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第10章 存储过程 10

存储过程把一个或多个T-SQL语句组合到一个逻辑单元中，在SQL Server数据库中保存为一个对象。在存储过程创建后，它的T-SQL定义保存在sys.sql_module系统目录视图中。

当存储过程首次执行的时候，SQL Server创建执行计划并把它存储在计划内存缓存中。然后SQL Server就能对这个存储过程的后续执行重用计划。和没有编译且没有准备好的等价即席查询相比，重用计划使得存储过程提供了快速可靠的性能。

注解 还可以创建使用.NET公共语言运行时（CLR）程序集的存储过程。我们会在第13章讨论。

本章包含了创建和管理存储过程的一些技巧。下面先介绍一下什么时候可以使用存储过程和它们会带来什么好处。

10.1 存储过程基础

这几年，我已经形成了一个只要有可能就使用存储过程的强烈偏好。根据我的经验，使用存储过程有很多好处，而没什么坏处。通常，反对使用存储过程的理由来自某些应用程序开发者，他们更喜欢在应用层使用即席SQL（adhoc SQL），并且没有训练过使用存储过程。与独立的应用程序和数据库管理员一起，存储过程同样面临着开发人员到数据库管理员对T-SQL代码的失控。假设你们的数据库管理团队很强，并且乐意及时提供向存储过程转移的帮助，那么使用存储过程的优势会比失去控制大很多。

使用存储过程有以下一些好处。

- 存储过程帮助在数据层聚集T-SQL代码。嵌入即席SQL的网站或应用程序在应用环境下很难修改，当即席SQL嵌入在应用程序内的时候，你可能会花费太多时间试图找到和调试嵌入的SQL。一旦找到了bug，你可能就需要重新编译可执行程序，引起不必要的应用程序临时停止或痛苦的应用程序部署。如果把T-SQL集中到存储过程中去，你就只需要集中在一个地方来查找SQL代码或SQL批处理。如果你能正确地为代码建立文档并对代码标准化，存储过程就会提升整个应用程序的可支持性。
- 存储过程帮助大的即席查询减少网络流量。编写应用程序调用而不是500行的SQL调用来执行存储过程，对网络以及应用程序的性能有正面影响，特别是当调用在一分钟内重复数千次时。
- 存储过程促进代码的可复用性。例如，如果你的网站应用程序使用一个下拉菜单来包含一组城市，并且这个下拉菜单用于很多网页，你可以在每个页面调用存储过程而不是在多个地方嵌入相同的SQL。
- 存储过程淡化数据获取的方法。如果你修改了提供源数据的基本表，存储过程（和视图相似）

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

能让应用程序对这个修改透明。这样就不需要修改应用程序底层的代码就能修改。你可以把老的表换成新的，而且只要同样的列和数据类型返回给应用程序，则应用程序完全不知情。

- 与视图不同，存储过程可以利用流控制技术、临时表、表变量等。
- 存储过程对查询响应时间的影响比较稳定。如果你使用大量的即席查询，可能会注意到有时候从查询中返回结果所花的时间变化很大。这可能是由诸如对表（锁）的并发活动或者资源问题（内存、CPU）等外部因素引起的。从另外一面来说，即席查询可能执行得不稳定，因为SQL Server有时候会选择效率较低的执行计划。而存储过程提供了更可靠的查询计划缓存，因此可以重用。注意到，这里我使用的是“可靠”这个单词，而不是“快速”。即席查询有时候确实比存储过程执行得更好，但是它完全依赖于执行计划被缓存的环境（“被发觉”的参数），以及看你怎么测试、调优以及实现代码了。

如果前面的这些理由还不能让你相信存储过程是非常好的，那么让我们再来看一下安全方面的好处。直接访问SQL Server实例和它的数据库的表（更坏的情况是访问sysadmin）会引起安全隐患。内联即席代码特别容易遭受SQL注入（injection）攻击。如果在T-SQL发送到SQL Server实例之前，有害的T-SQL就插入到了既有应用程序的T-SQL代码，那么就会发生SQL注入。除了SQL注入攻击，如果一些人得到了内联代码，他们就能收集数据库中基础架构的信息，从而指导他们尝试攻击。让所有SQL在存储过程内能保证只有存储过程被应用程序引用——而不是每一个列和表名。

存储过程的另外一个安全方面的好处就是可以授予数据库用户和/或数据库角色对它们的访问而不是授予对表的直接访问权限。存储过程能作为控制层，你可以选择哪些列和行能被存储过程（和调用者）修改，哪些不能。

10.1.1　创建基本的存储过程

存储过程能用于多种不同的活动，包括简单的SELECT、INSERT、UPDATE、DELETE等。本书中讨论的很多特性和语句都能在存储过程主体内使用。T-SQL活动能混合在单个存储过程中，或者你可以以
326 模块形式创建存储过程，为每一个或一组任务创建多个存储过程。

没有参数的存储过程的基本语法如下：

```
CREATE PROCEDURE [schema_name.] procedure_name
AS { <sql_statement> [ ...n ] }
```

命令的第一个参数是架构和新存储过程的名字。sql_statement参数是存储过程的T-SQL主体。这个参数包含了一个或多个你希望完成的任务。在这个实例中，演示了如何创建一个从AdventureWorks数据库查询数据的基本存储过程：

```
USE AdventureWorks
GO

CREATE PROCEDURE dbo.usp_SEL_ShoppingCartDisplay
AS

SELECT sc.ShoppingCartID,
       sc.ShoppingCartItemID,
       sc.Quantity,
       sc.ProductID,
       p.Name ProductName,
       p.ListPrice
FROM Sales.ShoppingCartItem sc
INNER JOIN Production.Product p ON
       sc.ProductID = p.ProductID
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
GO
```

然后，使用EXEC命令执行新的存储过程：

```
EXEC dbo.usp_SEL_ShoppingCartDisplay
```

这个查询返回如下的结果：

ShoppingCartID	ShoppingCartItemID	Quantity	ProductID	ProductName	ListPrice
14951	2	3	862	Full-Finger Gloves, M	37.99
20621	4	4	881	Short-Sleeve Classic Jersey, S	53.99
20621	5	7	874	Racing Socks, M	8.99

解析

在这个技巧中，演示了创建一个存储过程来查询两个表的内容并返回结果集。存储过程的工作方式和视图差不多，只是它第一次执行的时候有一个缓存查询计划，在连续执行中使之对运行时兼容。

这个示例从创建一个叫usp_SEL_ShoppingCartDisplay的存储过程开始：

```
CREATE PROCEDURE dbo.usp_SEL_ShoppingCartDisplay
AS
```

T-SQL查询定义跟在AS关键字后面：

```
SELECT sc.ShoppingCartID,
       sc.ShoppingCartItemID,
       sc.Quantity,
       sc.ProductID,
       p.Name ProductName,
       p.ListPrice
FROM Sales.ShoppingCartItem sc
INNER JOIN Production.Product p ON
       sc.ProductID = p.ProductID

GO
```

GO关键字用于标记存储过程的结束。

在本技巧的存储过程创建后，我们使用EXEC命令来执行：

```
EXEC dbo.usp_SEL_ShoppingCartDisplay
```

在创建存储过程的时候，SQL Server会检查SQL语法的正确性，但不会检查引用表是否存在。这意味着你可能引用了不正确的表名，并且直到运行时才会引发错误。这叫做延迟名称解析（deferred name resolution），使用它可以创建和引用数据库内不存在的对象。这也意味着可以删除、更改或修改存储过程中引用的对象，而无需先使其失效。

10.1.2 创建带参数的存储过程

在前面的技巧中，演示了不带参数的存储过程，也就是说没有传入外部参数。给它们传递参数的能力是存储过程为SQL Server中最重要的数据库对象的部分原因。使用参数，你可以把信息传递到存储过程的主体中来返回自定义的搜索信息，或使用参数来影响或执行对表的INSERT、UPDATE或DELETE语句。一个存储过程能包含多达2 100个参数（尽管不太可能使用这么多）。

创建存储过程的语法如下：

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
```

327

10

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        [ VARYING ] [ = default ] [ OUT | OUTPUT ] [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { <sql_statement> [;][ ...n ] | <method_specifier> }
```

参数以@符号作为前缀，后面是参数的数据类型和可选的默认值。参数有两类：输入和输出。输入参数用来把信息传递给存储过程进行处理，输出参数用于将信息返回给存储过程调用者。

在本例中，创建了一个新的、接受3个参数的存储过程。根据这些参数的值来更新表中的既有行或者插入新行：

```
CREATE PROCEDURE dbo.usp_UPD_ShoppingCartItem
(@ShoppingCartID nvarchar(50),
@Quantity int = 1, -- 默认数量为
@ProductID int)
AS
-- 如果参数传入的ShoppingCartID和ProductID相同，则更新数量

IF EXISTS(SELECT *
FROM Sales.ShoppingCartItem
WHERE ShoppingCartID = @ShoppingCartID AND
  ProductID = @ProductID )
BEGIN

     UPDATE Sales.ShoppingCartItem
     SET Quantity = @Quantity
     WHERE ShoppingCartID = @ShoppingCartID AND
       ProductID = @ProductID

     PRINT 'UPDATE performed. '
END
ELSE
BEGIN
     -- 否则插入新行
     INSERT Sales.ShoppingCartItem
     (ShoppingCartID, ProductID, Quantity)
     VALUES (@ShoppingCartID, @ProductID, @Quantity)

     PRINT 'INSERT performed. '
END

GO
```

328

接着，调用新的存储过程，为期望的参数传入3个值：

```
EXEC usp_UPD_ShoppingCartItem '1255', 2, 316
```

这个查询返回：

```
(1 行受影响)
INSERT performed.
```

解析

这个技巧演示了创建一个能接受参数的存储过程。在示例中，为存储过程定义了3个参数：

```
CREATE PROCEDURE usp_UPD_ShoppingCartItem
(@ShoppingCartID nvarchar(50),
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
@Quantity int = 1, -- 默认数据为
@ProductID int)
AS
```

第一个参数和第三个参数是必需的，都没有指定默认值。第二个参数是可选的，但它定义了@Quantity默认值为1。

存储过程的主体跟在AS关键字后面，从代码的第一部分开始，它通过IF语句检查行是否存在：

```
IF EXISTS(SELECT *
FROM Sales.ShoppingCartItem
WHERE ShoppingCartID = @ShoppingCartID AND
  ProductID = @ProductID )
BEGIN
```

如果对于某个ProductID和ShoppingCartID的行存在，数量会根据新的@Quantity值进行更新： 329

```
    UPDATE Sales.ShoppingCartItem
    SET Quantity = @Quantity
    WHERE ShoppingCartID = @ShoppingCartID AND
      ProductID = @ProductID

    PRINT 'UPDATE performed.
END
```

否则，如果这样的行不存在，将执行一个INSERT：

```
ELSE
BEGIN
    -- Otherwise insert a new row
    INSERT Sales.ShoppingCartItem
    (ShoppingCartID, ProductID, Quantity)
    VALUES (@ShoppingCartID, @ProductID, @Quantity)

    PRINT 'INSERT performed. '

END

GO
```

在存储过程创建后，执行存储过程并附上需要的参数值：

```
EXEC usp_UPD_ShoppingCartItem '1255', 2, 316
```

在这里，指定的ShoppingCartID和ProductID组合在表中不存在，所以向Sales.ShoppingCartItem表插入了一个新行。

10.1.3 使用 OUTPUT 参数

在前面的技巧中，我们看到了在存储过程定义中包含OUTPUT参数。OUTPUT参数把信息返回给存储过程调用者，无论它是另外一个存储过程调用还是应用程序进行的即席调用。

在这个示例中，创建了一个返回某个组的部门列表的存储过程。除了返回部门列表之外，还定义了一个OUTPUT参数来保存那个组的部门数：

```
CREATE PROCEDURE dbo.usp_SEL_Department
   @GroupName nvarchar(50),
   @DeptCount int OUTPUT
AS
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SELECT Name
FROM HumanResources.Department
WHERE GroupName = @GroupName
ORDER BY Name

SELECT @DeptCount = @@ROWCOUNT

GO
```

330 然后，调用新的存储过程。定义一个局部变量来保存OUTPUT参数值：

```
DECLARE @DeptCount int

EXEC dbo.usp_SEL_Department 'Executive General and Administration',
@DeptCount OUTPUT

PRINT @DeptCount
```

这个查询返回如下的结果集：

```
Name
Executive
Facilities and Maintenance
Finance
Human Resources
Information Services
```

除了结果之外，最终的行数也通过PRINT命令返回了：

```
5
```

解析

本技巧先创建了一个存储过程，其参数定义为@DeptCount，后面跟的是数据类型和OUTPUT关键字：

```
@DeptCount int OUTPUT
```

在存储过程的定义中，这个参数就根据在它前面执行的SELECT语句，被赋值为行总数值：

```
SELECT @DeptCount = @@ROWCOUNT
```

要在T-SQL代码中使用OUTPUT值，需要声明一个局部变量并在EXEC语句中使用：

```
DECLARE @DeptCount int
```

注意到，OUTPUT关键字跟在第二个参数后面，这是为了表明它是用来接收而不是发送真实值的：

```
EXEC dbo.usp_SEL_Department 'Executive General and Administration',
                            @DeptCount OUTPUT
```

可以将OUTPUT参数用作将信息返回给存储过程调用者的另一个方法。捕获OUTPUT结果可以再把变量值传给另外一个存储过程或进程。如果仅使用OUTPUT将信息返回给调用的应用程序，创建另一个结果集来保存需要的信息也同样容易。这是因为.NET应用程序（举例）能轻松处理从一个存储过程返回
331 的多个结果集。使用OUTPUT参数还是使用另一个结果集来返回信息只是个偏好问题。

10.1.4　修改存储过程

ALTER PROCEDURE命令用来修改存储过程的定义，使用它可以修改除原始存储过程名外的所有东西。语法和CREATE PROCEDURE非常相似。

在这个技巧中，会演示修改前面技巧中创建的既有存储过程，来使用独立的结果集返回部门数量，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

而不是使用OUTPUT参数：

```
ALTER PROCEDURE dbo.usp_SEL_Department
   @GroupName nvarchar(50)
AS

SELECT Name
FROM HumanResources.Department
WHERE GroupName = @GroupName
ORDER BY Name

SELECT @@ROWCOUNT DepartmentCount
GO
```

接着，执行修改后的存储过程：

```
EXEC dbo.usp_SEL_Department 'Research and Development'
```

这个查询返回两个结果集：

```
Name
Engineering
Research and Development
Tool Design
```

和

```
DepartmentCount
3
```

解析

在这个技巧中，用ALTER PROCEDURE来修改既有存储过程的定义——移除参数并且增加第二个结果集。使用这个命令，可以修改除过程名之外的所有东西。使用ALTER PROCEDURE也会保持存储过程上既有的权限，而不需要在更改后进行显式的重定义。

10.1.5 删除存储过程

可以使用DROP PROCEDURE命令从数据库删除存储过程。

删除存储过程的语法是：

```
DROP PROCEDURE { [ schema_name. ] procedure } [ ,...n ]
```

这个命令接受一个参数：要删除的存储过程名（一个或多个）。例如：

```
DROP PROCEDURE dbo.usp_SEL_Department
```

解析

一旦存储过程被删除，它的定义信息就从数据库的系统表中删除了。那个存储过程的任何缓存查询执行计划也被移除。一旦存储过程被删除，其他存储过程或者触发器引用这个存储过程的代码会失效。

10.1.6 在 SQL Server 启动时自动执行存储过程

在启动SQL Server服务的时候可以指定一个要执行的存储过程。你可能希望这样做来执行SQL Server实例需要的任何清空任务（例如，在服务启动时创建文档或清除工作表）。

存储过程的这种自动执行通过使用sp_procoption系统存储过程来实现。这个命令看起来像是接受

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

几个不同的选项，但在SQL Server中，它确实只执行一个任务，就是在SQL Server服务重新启动的时候设置存储过程自动执行。

在这个实例中，我们设置在SQL Server启动的时候存储过程自动执行。首先，把数据库上下文设置成master数据库（只有在这个地方才能放置自动执行的存储过程）：

```
USE master
GO
```

接着，（例如）创建一个启动日志表：

```
CREATE TABLE dbo.SQLStartupLog
(SQLStartupLogID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
 StartupDateTime datetime NOT NULL)
GO
```

现在，创建一个新的存储过程来向新表插入值（可以使用这个表来查看SQL Server什么时候重启动了）：

```
CREATE PROCEDURE dbo.usp_INS_TrackSQLStartups
AS

INSERT dbo.SQLStartupLog
(StartupDateTime)
VALUES (GETDATE())

GO
```

接着，使用sp_procoption存储过程来设置当SQL Server服务重启动的时候执行新的存储过程：

```
EXEC sp_procoption @ProcName = 'usp_INS_TrackSQLStartups',
  @OptionName = 'startup',
  @OptionValue = 'true
```

一旦服务重启动，表中就会插入一个新行。要再次禁用这个存储过程，需要执行下面的命令：

333

```
EXEC sp_procoption @ProcName = 'usp_INS_TrackSQLStartups',
  @OptionName = 'startup',
 @OptionValue = 'false'
```

解析

在这个技巧中，我们在master数据库中新建了一个表来跟踪SQL Server的启动。存储过程同样创建于master数据库，向表插入当前的日期和执行时间作为新行。

警告　我不赞成在系统数据库内创建对象。这通常不是一个好主意。但是如果你必须使用本技巧讨论的自动执行功能，你只能这么做（例如，如果你的IT部门为了跟踪目的需要SQL Server服务启动时间的日志）。

接着，调用sp_procoption来设置存储过程的启动值：

```
EXEC sp_procoption @ProcName = 'usp_INS_TrackSQLStartups',
  @OptionName = 'startup',
  @OptionValue = 'true'
```

在使用sp_procoption后，只要SQL Server服务重启动，一个新行就会插入到dbo.SQLStartupLog表。

存储过程必须创建在master数据库中，否则你尝试使用sp_procoption的时候就会看到如下的错误消息：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
消息15398，级别11，状态1，过程 sp_procoption，第73行
只有dbo拥有的master数据库中的对象才能更改启动设置。
```

10.1.7 报告存储过程元数据

可以使用sys.sql_modules目录视图来探究存储过程元数据（和其他对象类型），如在这个技巧中演示的：

```
SELECT definition,
     execute_as_principal_id,
     is_recompiled,
     uses_ansi_nulls,
     uses_quoted_identifier
FROM sys.sql_modules m
INNER JOIN sys.objects o ON
     m.object_id = o.object_id
WHERE o.type = 'P'
```

解析

sys.sql_modules视图用来查看存储过程、触发器、视图和其他SQL定义对象的定义和设置。在这个技巧中，sys.sql_modules联结到sys.objects，这样sys.objects中只有P类型（存储过程）的行才会被返回。

这个查询返回存储过程定义（如果没有加密）、EXECUTE AS安全上下文ID、存储过程是否设置了WITH RECOMPILE，以及当存储过程创建的时候如果ANSI NULL或QUOTED IDENTIFIER选项为ON返回的1。

提示 加密、EXECUTE AS和WITH RECOMPILE都会在本章讨论。 334

10.1.8 为存储过程建立文档

下面的技巧与对命令或函数的讨论相比更是最佳实践。恰当地注释你的存储过程代码很重要，这样，以后的支持部门、程序设计者和编辑才能理解业务规则和T-SQL代码所代表的意图。尽管在编写的时候一些代码看起来能“自说明”，但是写了几个月之后这些原始的逻辑可能就不是很清晰了。业务逻辑通常是短暂的，时间长了很难理解，因此将注释包含在代码的正文中往往会为排除故障和研究代码节约时间。

为了简单，本章的存储过程示例并没有包含大量的注释或头。然而，在正式的数据库中，至少应该为在正式数据库中创建的每个存储过程定义一个头。

下面是一个存储过程头的示例：

```
CREATE PROCEDURE dbo.usp_IMP_DWP_FactOrder
AS
------------------------------------------------------------
-- 目的：填充数据仓库，由作业调用
--
-- 维护日志
--
-- 维护人          维护时间    描述
-- -----------     ---------   --------------------------
-- Joe Sack        8/15/2008   创建
-- Joe Sack        8/16/2008   基础表增加了一列
-- 所以在这里也相应增加
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
... 这里是T-SQL
```

解析

这个示例演示了如何在新存储过程的主体内包含头信息。它跟踪存储过程的目的、调用它的应用程序以及维护日志。

警告　自书文档的缺点是编辑你代码的其他开发者可能没有把他们自己的更改包含到文档中。你会因为你没有写的代码而受到责备，只是因为你是最后记录更改的人。这就是公司要认真考虑源代码控制系统来跟踪所有签入和签出活动的原因，这个系统也可以比较各不同存储过程版本之间的变更。

毫无疑问，你也会在这个字段中看到其他头，它们包含了更多信息。尽管这样，我还是坚持不要过分要求过多的文档。只要能够说清楚就可以了，不要太多而显得啰嗦。例如，如果除了实际的CREATE PROCEDURE，你还在头中包含存储过程名，你不久就会在代码中看到头名和存储过程名不匹配。那为什么不只把那些存储过程定义中不包括的信息列入文档呢？

10.2　存储过程安全

335

在本章的开始我提到过，存储过程先天就有安全优势，现在再来复习一遍。

内联即席代码更容易受到SQL注入攻击，允许黑客看到嵌入的SQL调用，并搜索诸如“社会保险号”（Social Security Number）或“信用卡”（Credit Card）这样的单词。把SQL代码嵌入到存储过程中可以淡化外界对架构的影响。

同样，使用存储过程来替代直接表访问对基础表提供了更好的安全，可以控制如何进行修改和数据如何获取（都在列和行级别）。可以对用户授予对存储过程的执行权限来替代对表的访问权限。这也是对数据库服务器的唯一调用路径，因此任何偷窥因素都不会看到你的SELECT语句。

除了这些固有的优势之外（所有你需要做的仅仅是使用存储过程并从中获益），还有一些特性需要知道。下面的一些技巧告诉你如何加密存储过程，以使查询定义不被看到。

在这个技巧后，我们会讨论如何为存储过程定义一个自定义的安全上下文。

10.2.1　加密存储过程

和视图一样，存储过程的T-SQL定义也能在数据库中实现内容加密，这样就不能读取过程的定义。在后端使用SQL Server的软件开发商通常加密存储过程来防止客户端或竞争对手的篡改或逆向工程。如果你使用了加密，请确保保存了原始的T-SQL定义，因为以后不能再轻易地解密它（合法并且可靠的）。而且应该只在正式把它部署到生产环境之前才进行加密。

要加密存储过程，可以在新存储过程的名字后面指定WITH ENCRYPTION，如下面的示例演示的那样：

```
CREATE PROCEDURE dbo.usp_SEL_EmployeePayHistory
WITH ENCRYPTION
AS

SELECT EmployeeID, RateChangeDate, Rate, PayFrequency, ModifiedDate
FROM HumanResources.EmployeePayHistory

GO
```

一旦创建了WITH ENCRYPTION，就不能再看到过程的文本定义：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
--查看过程的文本
EXEC sp_helptext usp_SEL_EmployeePayHistory
```

这个查询返回：

```
对象'usp_SEL_EmployeePayHistory' 的文本已加密.
```

解析

加密能使用CREATE PROCEDURE或ALTER PROCEDURE来定义，但是要确保保存了你的源代码，因为既有的已加密文本不能被轻易解密。

336

10.2.2 使用 EXECUTE AS 来指定过程的安全上下文

WITH EXECUTE AS关键字允许你指定存储过程执行所在的安全上下文，覆盖存储过程调用者的默认安全。在这里，安全上下文是指用户执行存储过程的权限。

注解 这个技巧讨论的几个安全特性和概念在第17章和第18章也有介绍。

可以在以下几个权限下执行存储过程：

- 调用者的安全上下文；
- 过程的编写者或最终修改者；
- 指定的登录名（如果你有那个登录名的IMPERSONATE（模拟）权限）；
- 存储过程的拥有者。

首先，先简单介绍一下调用者权限和所有权链。当创建一个对象（例如存储过程或视图）并将其用来对另外一个数据库对象进行INSERT、UPDATE、DELETE或SELECT的对象的时候，就出现了*所有权链*。如果存储过程对象的架构和它引用对象的架构一样，SQL Server只会检测存储过程调用者是否有对存储过程的EXECUTE权限。再次说，所有权链仅仅应用于INSERT、UPDATE、DELETE或SELECT命令。可以授予用户执行存储过程的权限，而不用给他们对基础表的访问权限——这就是为什么存储过程对数据库的安全非常好。

然而，当发现执行的命令不是INSERT、UPDATE、DELETE或SELECT时，问题就来了。在这些情况下，即使调用者对存储过程有EXECUTE权限，例如使用TRUNCATE TABLE命令来截断表，他或她仍然先需要有使用TRUNCATE TABLE命令的权限。例如，创建下面的存储过程来删除表中的所有数据：

```
USE AdventureWorks
GO
CREATE PROCEDURE dbo.usp_DEL_ALLEmployeeSalary
AS
-- 在数据进来之前删除所有行
DELETE dbo.EmployeeSalary
GO
```

创建并填充dbo.EmployeeSalary表来设置这个环境：

```
CREATE TABLE dbo.EmployeeSalary
(EmployeeID int NOT NULL PRIMARY KEY CLUSTERED,
 SalaryAMT money NOT NULL)
 GO

INSERT dbo.EmployeeSalary (EmployeeID, SalaryAMT)
VALUES (1,45000.00), (343, 100000.00),(93, 3234993.00)
```

337

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

然后，将对新存储过程的EXECUTE权限授予你的雇员BrainG：

```
USE master
GO

CREATE LOGIN BrianG WITH PASSWORD = '1301C636F9D'

USE AdventureWorks
GO

CREATE USER BrianG
GO
GRANT EXEC ON usp_DEL_ALLEmployeeSalary to BrianG
```

现在，如果BrainG尝试执行这个过程，就建立了他的所有权链：

```
EXECUTE dbo.usp_DEL_ALLEmployeeSalary
```

虽然BrainG除了有执行这个新存储过程的权限之外在数据库中没有其他权限，但是他仍然成功了：

```
(3行受影响)
```

现在，把这个存储过程修改成使用TRUNCATE TABLE替代DELETE：

```
ALTER PROCEDURE dbo.usp_DEL_ALLEmployeeSalary
AS

-- 在数据进来之前删除所有行
TRUNCATE TABLE dbo.EmployeeSalary

GO
```

现在，如果BrainG再次尝试执行这个存储过程，SQL Server会检测BrainG使用TRUNCATE TABLE命令的能力，并返回如下错误（因为他只有执行过程的权限）：

```
消息1088，级别16，状态7，过程 usp_DEL_ALLEmployeeSalary，第5行
找不到对象"EmployeeSalary"，因为它不存在或者您没有所需的权限。
```

现在考虑一下为存储过程使用EXECUTE AS选项。使用EXECUTE AS，可以指定存储过程的任何调用者在你的安全上下文下运行。例如，假设前面的存储过程写成：

```
ALTER PROCEDURE dbo.usp_DEL_ALLEmployeeSalary
WITH EXECUTE AS OWNER
AS

-- 在数据进来之前删除所有行
TRUNCATE TABLE dbo.EmployeeSalary

GO
```

加了WITH EXECUTE AS OWER后，BrianG只需要有对存储过程的EXECUTE权限，就能在拥有者的安全
338 上下文下执行存储过程。如果拥有者有TRUNCATE表的权限，那么存储过程就会成功执行。

对于动态SQL也要说一下。即使动态SQL执行的是INSERT、UPDATE、DELETE或SELECT。SQL Server会确保调用者有EXECUTE权限和要执行动态SQL要进行的操作所需要的合适权限。

例如，下面的过程包含动态的SQL，使用它可以根据@SchemaAndTable输入参数指定的架构名和表名来选择任何表的行总数：

```
CREATE PROCEDURE dbo.usp_SEL_CountRowsFromAnyTable
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
@SchemaAndTable nvarchar(255)
AS

EXEC ('SELECT COUNT(*) FROM ' + @SchemaAndTable)

GO
```

如果有权限执行存储过程并且能访问指定的表，SQL Server会允许返回行数：

```
EXEC dbo.usp_SEL_CountRowsFromAnyTable 'HumanResources.Department'
```

这个查询返回：

```
17
```

然而，授予EXECUTE权限是不够的。因为它是动态的SQL，如果用户对基础表没有SELECT权限，SQL Server会检测对存储过程的EXECUTE权限和对表的SELECT权限。如果用户BrianG没有SELECT权限，他会看到下面的错误：

```
消息229，级别14，状态5，第1行
拒绝了对对象'Department'、数据库'AdventureWorks'、架构'HumanResources'的SELECT权限。
```

同样，对于这种情况可以使用EXECUTE AS来补救（当然，如果你希望BrianG有这些权限）。这次，我们会指定一个显式用户名作为这个过程的安全上下文（我还创建了SteveP用户，它有HumanResources.Employee表的读权限）：

```
USE master
GO

CREATE LOGIN SteveP WITH PASSWORD = '533B295A-D1F0'

USE AdventureWorks
GO

CREATE USER SteveP

GRANT SELECT ON OBJECT::HumanResources.Employee TO SteveP
GO

ALTER PROCEDURE dbo.usp_SEL_CountRowsFromAnyTable
   @SchemaAndTable nvarchar(255)
WITH EXECUTE AS 'SteveP'
AS

-- 可以用在任何SteveP可以SELECT的表上
EXEC ('SELECT COUNT(*) FROM ' + @SchemaAndTable)
GO
```

假设SteveP对作为过程中的动态SQL传递的任何表都有适当的权限，现在如果BrianG有执行这个过程的权限并执行它，他会看到返回的结果，就好像BrianG是SteveP一样。SQL Server不会检测BrianG的权限，而是使用SteveP的安全上下文。

解析

在这个技巧中，演示了在存储过程中使用EXECUTE AS，它可以定义存储过程执行的安全上下文，而不管调用者是什么。

存储过程中的EXECUTE AS选项如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
EXECUTE AS { CALLER | SELF | OWNER | 'user_name' }
```

EXECUTE AS的默认行为是CALLER选项，它代表了使用执行用户的权限（如果用户没有合适的权限，执行会失败）。如果使用了SELF选项，存储过程的执行上下文将是创建或上次更新存储过程的用户。如果指定了OWNER选项，则使用存储过程的架构的拥有者。user_name选项是对数据库用户的显式引用，存储过程会在他的安全上下文中执行。

10.3 重新编译和缓存

存储过程能提供性能优势是因为缓存的查询执行计划，允许SQL Server来重用既有的计划而不是生成一个新的。存储过程也能稳定查询响应时间，与之相比，即席查询的响应时间变化很大。

注解 更多有关评估查询性能和过程缓存的内容参阅第28章。

虽然这么说，但存储过程并不是提升查询性能的万能药。你仍然需要评估存储过程主体内各条语句的性能，并确保表已经被正确索引以及数据库的设计是高效率的。本书的其他章节讨论的一些特性可以用在存储过程的主体中，但是即使在存储过程外使用，你也必须进行这样的考虑。

在下面的两个技巧中，我会讨论你不希望缓存查询计划的情况。第一个介绍了RECOMPILE选项，第
340 二个介绍了DBCC FREEPROCCACHE命令。

10.3.1 每次存储过程执行时重新编译

当存储过程的计划自动或显式重建的时候就会发生重新编译。当存储过程中引用的基础表或其他对象发生改变后，存储过程就会在其执行期间自动重新编译。计划使用的索引发生变动或者存储过程引用的表键发生了大量的更新也可能引起重新编译。自动重新编译的目的是确保SQL Server执行计划使用最新的信息，不使用过期的假设架构和数据。

SQL Server在存储过程中使用语句级别的重新编译，而不是重新编译整个存储过程。重新编译在生成新计划的时候会引起额外的负载，而语句级别的重新编译能减小这种负载，因为它只纠正需要纠正的地方。

尽管重新编译代价很高，并且应该在大多数时候避免，但是有时候你的存储过程还是有需要强制重新编译的理由。例如，由于限定列不同的选择性，存储过程可能基于调用它的应用程序生成差异很大的查询结果——以至于保留的执行计划在使用不同的输入参数时会引起性能问题。

例如，如果一个参数指定的城市返回一百万行，而另外一个城市值返回一行，SQL Server可能就没必要缓存正确的执行计划了。可能最终缓存的计划是为一行优化的而不是一百万行，这样会引起很长的查询执行时间。如果你觉得除了缓存之外存储过程还在其他地方有用，就可以使用WITH RECOMPILE命令。

在这个示例中，演示了如何强制存储过程在每次执行的时候重新编译：

```
CREATE PROCEDURE dbo.usp_SEL_BackupMBsPerSecond
(@BackupStartDate datetime,
 @BackupFinishDate datetime)
WITH RECOMPILE -- 从不保存计划
AS

-- 衡量db备份吞吐量
SELECT (SUM(backup_size)/1024)/1024 as 'MB',
       DATEDIFF ( ss , MIN(backup_start_date),
       MAX(backup_finish_date)) as 'seconds',
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    ((SUM(backup_size)/1024)/1024 )/
        DATEDIFF ( ss , MIN(backup_start_date) ,
            MAX(backup_finish_date)) as 'MB per second'
FROM msdb.dbo.backupset
WHERE backup_start_date >= @BackupStartDate AND
backup_finish_date < @BackupFinishDate AND
type = 'd'
GO
```

现在，只要这个过程被调用，SQL Server就会形成新的执行计划。

解析

这个过程使用WITH RECOMPILE来确保过程创建和执行的时候查询计划不被缓存。

可以确信，你只会在很少的环境下使用WITH RECOMPILE，因为通常情况下，SQL Server选择的缓存计划就能满足要求。如果你仍然希望利用存储过程的其他优势（比如安全和模块化），但又不希望SQL Server来为变化很大的结果集保存一个低效的执行计划（“参数发觉”），就可以使用这个选项。 341

提示 更多关于查询执行计划的信息请参阅第28章。

10.3.2 清空过程缓存

在这个技巧中，会演示如何从存储过程缓存中移除所有计划。这个技术通常用于测试没有缓存下的存储过程性能，重新生成缓存就好像SQL Server刚重启动一样。如果你不希望既有的缓存查询计划对存储过程性能测试产生影响，这是开发SQL Server实例的一种选项。

警告 不要在正式环境下使用这个命令，因为可能会删除一些工作得很好的缓存查询计划。

在这个示例中，在执行DBCC FREEPROCCACHE之前先统计缓存查询计划的总数：

```
SELECT COUNT(*) 'CachedPlansBefore'
FROM sys.dm_exec_cached_plans
```

这个查询返回：

```
CachedPlansBefore
42
```

接着，整个SQL Server实例的过程缓存被清空：

```
DBCC FREEPROCCACHE

SELECT COUNT(*) 'CachedPlansAfter'
FROM sys.dm_exec_cached_plans
```

然后，再一次执行第一次的查询来看看缓存计划的个数：

```
DBCC 执行完毕。如果DBCC 输出了错误信息，请与系统管理员联系。
CachedPlansAfter
----------------
0

(1 行受影响)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

这个技巧中使用了DBCC FREEPROCCACHE来清空过程缓存。如果你自己尝试的话，已缓存的计划个数会根据你的SQL Server实例的行为不尽相同。它包括清空缓存前后运行的任何后台进程或作业。sys.dm_exec_cached_plans动态管理视图用于演示DBCC命令的作用，显示了原来的42个计划清空后为0（基于你的SQL Server实例上正在进行的活动，你的结果可能会不同。）

342

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第11章 用户定义函数和类型

在本章中，会介绍一些关于用户定义函数和类型的技巧。用户定义函数(UDF)允许我们把逻辑和子程序封装到一个函数中，然后就可以在T-SQL查询和可编程对象中使用它们。用户定义类型（UDT）允许我们根据底层系统数据底层来创建别名类型，强制一个特殊的数据类型、长度和可空性。

本章最后，还要复习SQL Server 2008用户定义表类型，可以在T-SQL代码中将它作为用户定义表参数使用，用于传递表结果集。

注解 本章会介绍如何使用T-SQL创建用户定义函数和类型。然而，第13章会简单介绍如何使用新的公共语言运行时（CLR）功能创建这些对象。在SQL Server 2008中，基于CLR的UDT不再限制到8 000字节了。

11.1 UDF基础

T-SQL用户定义函数（UDF）分成3类：标量（scalar）、内联表值（inline table-valued）和多语句表值（multi-statement table-valued）。

标量用户定义函数用于根据0个或多个参数返回一个值。例如，你可以创建一个接受CountryID作为参数的标量UDF，并返回CountryNM。

警告 如果在SELECT子句中使用标量用户定义函数，函数将为FROM子句中每一行执行，根据函数的设计情况，这样很可能会造成很差的性能。

内联表值UDF根据用于定义返回的行和列的一个SELECT语句来返回表数据类型。和存储过程不同，内联UDF能在查询的FROM子句中引用，还能联结到其他表。和视图不同，内联UDF能接受参数。

多语句表值UDF也返回结果集并能在FROM子句中引用。和内联表值UDF不同，它们不限制在函数定义中只能使用一个SELECT语句，而允许在UDF定义的主体中有多个T-SQL语句，以定义一个返回的最终结果集。

UDF还能用在存储过程不能使用的地方，比如查询的FROM和SELECT子句中。UDF也有助于代码重用。例如，如果创建一个标量UDF，它根据ConutryID返回ConutryNM，而这种函数在几个不同的存储过程中都需要，可以直接调用UDF函数而不是重复20行代码来完成这个查找操作。

在下面的几个技巧中，会演示如何创建、删除、修改和查看这些UDF类型的元数据。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

11.1.1　创建标量用户定义函数

标量用户定义函数接受0个或多个参数并返回一个值，标量UDF通常用于将当前的值转换为新值，或者根据某些参数执行其他复杂的查找。标量函数能用于搜索、列以及联结表达式。

标量UDF简化的语法如下：

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
    [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
                function_body
       RETURN scalar_expression
END
```

注解　CREATE FUNCTION的完整语法可以参看SQL Server的联机丛书

表11-1中简单介绍了每个参数的使用。

表11-1　标量UDF参数

参　数	描　述
[schema_name.] function_name	可选的架构名和必需的新标量UDF的函数名
@parameter_name	要传给UDF的参数名，必须以@符号作为前缀
[type_schema_name.] scalar_parameter_data_type	可选的参数数据类型架构和相关联的参数数据类型
[,...n]	尽管不是实际的参数，这个语法元素表示能定义一个或多个参数（最多1 024个）
return_data_type	这个参数指定了用户定义函数要返回的数据类型
function_body	函数体包含一个或多个用于产生和计算标量值的T-SQL语句
scalar_expression	将被标量函数返回的真实值（注意，它是定义在函数体的后面）

344

本例创建了一个接受一个varchar(max)数据类型的参数的标量UDF。它根据传入的参数是否包含可疑值（由函数定义）来返回一个bit值（1或0）。因此如果输入参数包含诸如DELETE或SHUTDOWN之类的命令，标志位就被设置成1：

```
-- 创建函数来检测应用程序的任何可疑的行为
CREATE FUNCTION dbo.udf_CheckForSQLInjection
   (@TSQLString varchar(max))
RETURNS BIT
AS

BEGIN

DECLARE @IsSuspect bit

-- UDF让字符串最左边填补一个空格
SET @TSQLString = ' ' + @TSQLString
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
IF    (PATINDEX('% xp_%' , @TSQLString ) <> 0 OR
   PATINDEX('% sp_%' , @TSQLString ) <> 0 OR
   PATINDEX('% DROP %' , @TSQLString ) <> 0 OR
   PATINDEX('% GO %' , @TSQLString ) <> 0 OR
   PATINDEX('% INSERT %' , @TSQLString ) <> 0 OR
   PATINDEX('% UPDATE %' , @TSQLString ) <> 0 OR
   PATINDEX('% DBCC %' , @TSQLString ) <> 0 OR
   PATINDEX('% SHUTDOWN %' , @TSQLString )<> 0 OR
   PATINDEX('% ALTER %' , @TSQLString )<> 0 OR
   PATINDEX('% CREATE %' , @TSQLString ) <> 0OR
   PATINDEX('%;%' , @TSQLString )<> 0 OR
   PATINDEX('% EXECUTE %' , @TSQLString )<> 0 OR
   PATINDEX('% BREAK %' , @TSQLString )<> 0 OR
   PATINDEX('% BEGIN %' , @TSQLString )<> 0 OR
   PATINDEX('% CHECKPOINT %' , @TSQLString )<> 0 OR
   PATINDEX('% BREAK %' , @TSQLString )<> 0 OR
   PATINDEX('% COMMIT %' , @TSQLString )<> 0 OR
   PATINDEX('% TRANSACTION %' , @TSQLString )<> 0 OR
   PATINDEX('% CURSOR %' , @TSQLString )<> 0 OR
   PATINDEX('% GRANT %' , @TSQLString )<> 0 OR
   PATINDEX('% DENY %' , @TSQLString )<> 0 OR
   PATINDEX('% ESCAPE %' , @TSQLString )<> 0 OR
   PATINDEX('% WHILE %' , @TSQLString )<> 0 OR
   PATINDEX('% OPENDATASOURCE %' , @TSQLString )<> 0 OR
   PATINDEX('% OPENQUERY %' , @TSQLString )<> 0 OR
   PATINDEX('% OPENROWSET %' , @TSQLString )<> 0 OR
   PATINDEX('% EXEC %' , @TSQLString )<> 0)

BEGIN
   SELECT @IsSuspect = 1
END
ELSE
BEGIN
   SELECT @IsSuspect = 0
END
   RETURN (@IsSuspect)
END

GO
```

345

然后，你应该通过计算3个不同的字符串输入值来测试这个函数。第一个包含了SELECT语句：

```
SELECT dbo.udf_CheckForSQLInjection
('SELECT * FROM HumanResources.Department')
```

这个查询返回：

```
0
```

下一个字符串包含了SHUTDOWN命令：

```
SELECT dbo.udf_CheckForSQLInjection
(';SHUTDOWN')
```

这个查询返回：

```
1
```

最后测试的字符串包含了DROP命令：

```
SELECT dbo.udf_CheckForSQLInjection
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
('DROP HumanResources.Department')
```

这个查询返回：

```
1
```

在下一个示例中，将创建可设置字符串大小写的用户定义函数：

```
CREATE FUNCTION dbo.udf_ProperCase(@UnCased varchar(max))
RETURNS varchar(max)
AS
BEGIN

   SET @UnCased = LOWER(@UnCased)

   DECLARE @C int
   SET @C = ASCII('a')

   WHILE @C <= ASCII('z')
   BEGIN

  SET @UnCased = REPLACE( @UnCased, ' ' + CHAR(@C), ' ' + CHAR(@C-32))
      SET @C = @C + 1
  END

346   SET @UnCased = CHAR(ASCII(LEFT(@UnCased, 1))-32) + RIGHT(@UnCased,
LEN(@UnCased)-1)

   RETURN @UnCased
END

GO
```

用户定义函数创建好后，将要修改大小写的字符串可用作函数的参数：

```
SELECT dbo.udf_ProperCase(DocumentSummary)
FROM Production.Document
WHERE FileName = 'Installing Replacement Pedals.doc'
```

这个查询返回：

```
Detailed Instructions For Replacing Pedals With Adventure Works Cycles Replacement
Pedals. Instructions Are Applicable To All Adventure Works Cycles Bicycle Models
And Replacement Pedals. Use Only Adventure Works Cycles Parts When Replacing Worn Or
Broken Components.
```

解析

这个技巧演示了一个标量UDF，在这个例子中接收一个参数并返回一个值。能在T-SQL代码中使用标量函数的一些地方包括：

- SELECT或GROUP BY子句中的列表达式；
- FROM子句中JOIN的搜索条件；
- WHERE或HAVING子句的搜索条件。

这个技巧从定义一个UDF名和参数开始：

```
CREATE FUNCTION dbo.udf_CheckForSQLInjection
   (@TSQLString varchar(max))
```

@TSQLString参数保存了要计算的varchar(max)字符串。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

代码的下一行把scalar_return_data_type定义为bit。这意味着由这个函数返回的一个值是bit数据类型：

```
RETURNS BIT
AS
```

BEGIN标志了function_body的开始，在这里形成了返回这个bit值的逻辑：

```
BEGIN
```

创建了一个局部变量来保存bit值。它是最后会被作为函数输出传递的参数：

```
DECLARE @IsSuspect bit
```

然后，把传入到UDF的字符串和一个空格进行拼接：

```
-- UDF让字符串最左边填补一个空格
SET @TSQLString = ' ' + @TSQLString
```

@TSQLString左边被填充了一个空格，这样搜索可疑单词或匹配的时候会简单一些。例如，如果可疑单词是在@TSQLString的开始，并且我们在搜索单词DROP，那么你必须使用PATINDEX来搜索'%DROP %'和'% DROP %'。毫无疑问，搜索'%DROP %'会给你诸如单词gumdrop这样的错误答案，所以你应该在字符串前面加上空格来避免混淆。

347

在IF语句中，使用PATINDEX来计算@TSQLString。对于每次计算，如果找到了匹配，条件会计算为TRUE：

```
IF    (PATINDEX('% xp_%' , @TSQLString ) <> 0 OR
   PATINDEX('% sp_%' , @TSQLString ) <> 0 OR
   PATINDEX('% DROP %' , @TSQLString ) <> 0 OR
   PATINDEX('% GO %' , @TSQLString ) <> 0 OR
   PATINDEX('% BREAK %' , @TSQLString )<> 0 OR
...
```

一旦任何一个条件计算为TRUE，@IsSuspect比特标志位就被设置为1：

```
BEGIN
   SELECT @IsSuspect = 1
END
ELSE
BEGIN
   SELECT @IsSuspect = 0
END
```

使用RETURN关键字来把@IsSuspect变量的标量值返回给调用者：

```
RETURN (@IsSuspect)
```

然后使用END关键字来结束UDF，并使用GO来结束批处理：

```
END

GO
```

然后使用新的标量UDF来检测3个不同的字符串值。第一个字符串SELECT * FROM HumanResources.Department被认为是干净的，但是后两个字符串;SHUTDOWN和DROP HumanResources.Department都返回了比特值1，因为它们与函数IF子句中的可疑单词搜索相匹配。

SQL Server没有内置的转换大小写的函数，因此在第二个示例中，我演示创建一个用户定义函数来执行这个操作。第一行的CREATE FUNCTION定义了参数的名称——在这个例子中为varchar(max)数据

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

类型参数：

```
CREATE FUNCTION dbo.udf_ProperCase(@UnCased varchar(max))
```

RETURNS关键字定义了应用逻辑之后函数返回什么数据类型：

```
RETURNS varchar(max)
AS
BEGIN
```

接着，先使用LOWER函数将函数传入的变量转换为小写：

348

```
SET @UnCased = LOWER(@UnCased)
```

再将新的整型局部变量@C设为字母a的ASCII值：

```
DECLARE @C int
SET @C = ASCII('a')
```

WHILE循环将遍历字母表中的每个字母，查找空格后面的字母，然后用对应的大写字母替换所有出现在空格之后的字母：

```
WHILE @C <= ASCII('z')
   BEGIN
   SET @UnCased = REPLACE( @UnCased, ' ' + CHAR(@C), ' ' + CHAR(@C-32))
      SET @C = @C + 1
END
```

小写字母的ASCII整数值减去32后就能转换成对应的大写字母。例如，小写字母a的ASCII值为97，对应的大写字母为65。

```
SET @UnCased = CHAR(ASCII(LEFT(@UnCased, 1))-32) + RIGHT(@UnCased, LEN(@UnCased)-1)
```

然后函数返回经过大小写转换后的@UnCased字符串值：

```
   RETURN @UnCased
END
GO
```

下一步，在查询的SELECT子句中，使用新的标量UDF将DocumentSummary文本转换大小写：

```
SELECT dbo.udf_ProperCase(DocumentSummary)
...
```

11.1.2　创建内联用户定义函数

内联UDF返回表数据类型。在UDF定义中，你不需要显式定义返回的表，只需要使用一个SELECT语句来定义返回的行和列。内联UDF使用一个或多个参数并使用一个SELECT语句返回数据。内联UDF和视图非常相似，它们能在FROM子句中引用。然而和视图不同的是，UDF可以接受参数，然后将它们用在函数的SELECT语句中。

基本语法如下：

```
CREATE FUNCTION [ schema_name. ] function_name
        ( [ { @parameter_name [ AS ]
[ type_schema_name. ] scalar_parameter_data_type [ = default ]
          } [ ,...n ]
       ]
        )
RETURNS TABLE
[ AS ]
RETURN [ ( ] select_stmt [ ) ]
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注解 CREATE FUNCTION的完整语法可参见SQL Server联机丛书。

表11-2详细描述了这个命令的参数。

349

表11-2 内联UDF参数

参 数	描 述
[schema_name.] function_name	这个参数定义了可选的架构名和必需的新内联UDF的函数名
@parameter_name	要传给UDF的参数名，必须以@符号作为前缀
[type_schema_name.] scalar_parameter_data_type	@parameter_name的数据类型和可选的拥有者架构（使用用户定义类型作为参数的时候用）
[,...n]	尽管不是实际的参数，这个语法元素表示能定义一个或多个参数（最多1 024个）
select_stmt	内联UDF返回的一个SELECT语句

这个示例演示了创建内联表UDF，它接受一个@EmployeeID整型参数并返回相关业务实体的地址：

```
CREATE FUNCTION dbo.udf_ReturnAddress
   ( @BusinessEntityID int)
RETURNS TABLE
   AS
RETURN (
        SELECT t.Name AddressTypeNM,
               a.AddressLine1,
               a.City,
               a.StateProvinceID,
               a.PostalCode
          FROM Person.Address a
      INNER JOIN Person.BusinessEntityAddress e ON
          a.AddressID = e.AddressID
      INNER JOIN Person.AddressType t ON
          e.AddressTypeID = t.AddressTypeID
      WHERE e.BusinessEntityID = @BusinessEntityID )

GO
```

然后，使用查询测试这个新函数，查询的FROM子句中引用了业务实体332：

```
SELECT AddressTypeNM, AddressLine1, City, PostalCode
FROM dbo.udf_ReturnAddress(332)
```

这个查询返回：

```
AddressTypeNM       AddressLine1              City          PostalCode
Shipping            26910 Indela Road         Montreal      H1Y 2H5
Main Office         25981 College Street      Montreal      H1Y 2H5
```

解析

在这个技巧中，创建了一个内联表UDF，且根据传入的@BusinessEntityID值获取业务实体的地址。这个UDF开始时和标量UDF类似，只是RETURNS命令使用TABLE数据类型（这是和标量UDF的区别）：

350

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE FUNCTION dbo.udf_ReturnAddress
   ( @BusinessEntityID int)
RETURNS TABLE
   AS
```

在AS关键字后发起了RETURN语句，圆括号中是一个SELECT语句：

```
RETURN (
        SELECT t.Name AddressTypeNM,
               a.AddressLine1,
               a.City,
               a.StateProvinceID,
               a.PostalCode
         FROM Person.Address a
      INNER JOIN Person.BusinessEntityAddress e ON
           a.AddressID = e.AddressID
      INNER JOIN Person.AddressType t ON
           e.AddressTypeID = t.AddressTypeID
      WHERE e.BusinessEntityID = @BusinessEntityID )

GO
```

在函数创建后，我们在SELECT查询的FROM子句中使用这个新的内联UDF。@BusinessEntityID值332传入了函数的圆括号中：

```
SELECT AddressTypeNM, AddressLine1, City, PostalCode
FROM dbo.udf_ReturnAddress(332)
```

这个函数然后返回了一个结果集，就好像查询视图或表一样。而且，和视图或存储过程相似，定义函数所创建的查询必须像普通SELECT语句那样调优。使用内联UDF不会比使用视图或存储过程有更多的性能优势。

11.1.3　创建多语句用户定义函数

多语句表UDF能像内联UDF一样在FROM子句中被引用，但是和内联UDF不同的是，它们不限制在函数定义内只能使用一个SELECT语句。多语句UDF能在UDF定义的主体内使用多个T-SQL语句以定义最后要返回的一个结果集。

多语句表UDF的基本语法如下：

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
    [ = default ] [READONLY] }
    [ ,...n ]
  ]
)
RETURNS @return_variable TABLE <table_type_definition>
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
                function_body
        RETURN
END
```

351

表11-3描述了这个命令的参数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表11-3 多语句UDF参数

参 数	描 述
[schema_name.] function_name	这个参数指定了可选的架构名和必需的新内联UDF的函数名
@parameter_name	要传给UDF的参数名，必须以@符号作为前缀
[type_schema_name.] scalar_parameter_data_type	@parameter_name的数据类型和可选的拥有者架构(使用用户定义类型作为参数的时候用)
[,...n]	尽管不是实际的参数，这个语法元素表示能定义一个或多个参数（最多1 024个）
@return_variable	用户定义的表变量名，它保存了UDF返回的结果
< table_type_definition >	这个参数包含了表变量一个或多个列的定义。每个列的定义包含了名字和数据类型，并且可以有选择地定义PRIMARY KEY、UNIQUE、NULL或CHECK约束
function_body	这个函数体包含了用于填充和修改UDF返回的表变量的一个或多个T-SQL语句

注意RETURNS关键字，它定义了表变量（table variable）的定义。同样地，注意函数最后的RETURN关键字，没有任何参数或查询在它之后，这样会返回定义的表变量。

在本例中，我们创建的多语句UDF接受两个参数：一个用来保存字符串，而另外一个定义字符串如何被分隔。字符串然后根据定义的分隔符分割成一个结果集：

```
-- 创建一个把字符串数组作为表结果集返回的UDF
CREATE FUNCTION dbo.udf_ParseArray
   ( @StringArray varchar(max),
     @Delimiter char(1) )
RETURNS @StringArrayTable TABLE (Val varchar(50))
AS
BEGIN
   DECLARE @Delimiter_position int

   IF RIGHT(@StringArray,1) != @Delimiter
   SET @StringArray = @StringArray + @Delimiter

   WHILE CHARINDEX(@Delimiter, @StringArray) <> 0
    BEGIN
      SELECT @Delimiter_position =
         CHARINDEX(@Delimiter, @StringArray)

      INSERT @StringArrayTable
      VALUES (left(@StringArray, @Delimiter_position - 1))

      SELECT @StringArray = STUFF(@StringArray, 1,
@Delimiter_position, '')
      END
RETURN
END

GO
```

现在，用它来分割一个以逗号分隔的数组值：

```
SELECT Val
FROM dbo.udf_ParseArray('A,B,C,D,E,F,G', ',')
```

这个查询返回下面的结果：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Val
A
B
C
D
E
F
G
```

解析

本技巧中创建的多语句表UDF使用了两个参数，第一个用来保存字符串，第二个用来定义分隔字符串的字符：

```
CREATE FUNCTION dbo.udf_ParseArray
   ( @StringArray varchar(max),
     @Delimiter char(1) )
```

接着，在RETURNS标记后定义一个表变量。@StringArrayTable用于保存分割成独立值后的字符串数组值：

```
RETURNS @StringArrayTable TABLE (Val varchar(50))
```

函数体从AS和BEGIN后开始：

```
AS
BEGIN
```

创建一个局部变量来保存字符串中的分隔符位置：

```
DECLARE @Delimiter_position int
```

如果字符串数组的最后一个字符不是分隔符值，就把分隔符值拼接到字符串数组的最后：

```
IF RIGHT(@StringArray,1) != @Delimiter
SET @StringArray = @StringArray + @Delimiter
```

创建一个WHILE循环，直到字符串数组中没有分隔符才停止循环：

```
WHILE CHARINDEX(@Delimiter, @StringArray) <> 0
   BEGIN
```

在循环中，使用CHARINDEX来标识分隔符的位置：

```
SELECT @Delimiter_position =
   CHARINDEX(@Delimiter, @StringArray)
```

353

LEFT函数和分隔符位置配合来把独立的分隔后的部分字符串放到表变量中：

```
INSERT @StringArrayTable
VALUES (left(@StringArray, @Delimiter_position - 1))
```

然后使用STUFF函数把插入的部分从字符串数组中移除：

```
SELECT @StringArray = STUFF(@StringArray, 1, @Delimiter_position, '')
```

STUFF用于删除一组字符并用另外的字符串替代。STUFF函数的第一个参数是字符表达式，在本例中就是字符串数组。第二个参数是要删除和插入文本的起始位置，在这里要移除的文本是从字符串的起始位置开始到第一个分隔符结束。第三个参数是要删除的字符长度，在本例中就是分隔符位置变量的值。最后一个参数是要插入的字符串，在这里是用两个单引号表示的空字符串——效果就像是第一个条目被删除了那样。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

插入值的过程会一直继续，直到字符串数组中不再有分隔符。之后，WHILE循环结束，调用RETURN返回表变量结果集。

```
      END
RETURN
END
GO
```

然后在FROM子句中引用新的UDF。UDF的第一个参数是以逗号分隔的字母列表。第二个参数是分隔符参数（一个逗号）：

```
-- 现在用它来分离逗号分隔的数组
SELECT Val
FROM dbo.udf_ParseArray('A,B,C,D,E,F,G', ',')
```

列表被分成一个结果集，每一个字母自占一行。可以看到，多语句表UDF比只能用一个SELECT语句的内联表值函数在编程上更复杂。

11.1.4 修改用户定义函数

使用ALTER FUNCTION命令能修改函数，如下面的技巧演示：

```
ALTER FUNCTION dbo.udf_ParseArray
   ( @StringArray varchar(max),
 @Delimiter char(1) ,
 @MinRowSelect int,
 @MaxRowSelect int)
RETURNS @StringArrayTable TABLE (RowNum int IDENTITY(1,1), Val
varchar(50))
AS
BEGIN
   DECLARE @Delimiter_position int

   IF RIGHT(@StringArray,1) != @Delimiter
   SET @StringArray = @StringArray + @Delimiter

   WHILE CHARINDEX(@Delimiter, @StringArray) <> 0
      BEGIN
      SELECT @Delimiter_position =
         CHARINDEX(@Delimiter, @StringArray)

      INSERT @StringArrayTable
      VALUES (left(@StringArray, @Delimiter_position - 1))

      SELECT @StringArray = stuff(@StringArray, 1,
@Delimiter_position, '')
      END

DELETE @StringArrayTable
WHERE RowNum < @MinRowSelect OR
  RowNum > @MaxRowSelect

RETURN
END

GO

-- 现在用它分离逗号分隔的数组
SELECT RowNum, Val
FROM dbo.udf_ParseArray('A,B,C,D,E,F,G', ',', 3, 5)
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
RowNum          Val
3               C
4               D
5               E
```

解析

ALTER FUNCTION允许以和CREATE FUNCTION差不多的语法来修改既有的UDF，但是有下面一些限制。

- 不能使用ALTER FUNCTION修改函数名。你能做的仅仅是替换既有函数的代码——所以函数需要事先存在。
- 不能将标量UDF转换为表UDF（内联或多语句），也不可以将表UDF转换为标量UDF。

在这个技巧中，修改前面技巧中的udf_ParseArray来增加@MinRowSelect和@MaxRow-Select两个参数：

```
ALTER FUNCTION dbo.udf_ParseArray
   ( @StringArray varchar(max),
     @Delimiter char(1) ,
      @MinRowSelect int,
      @MaxRowSelect int)
```

@StringArrayTable表变量也加入了一个叫RowNum的列，该列设置了IDENTITY属性（也就是说，对于结果集中的每行会增加一个整数值）：

355

```
RETURNS @StringArrayTable TABLE (RowNum int IDENTITY(1,1), Val varchar(50))
```

其他的修改在WHILE循环结束后。任何小于最小值或者大于最大值的RowNum值都会从@StringArrayTable表数组中删除：

```
DELETE @StringArrayTable
WHERE RowNum < @MinRowSelect OR
   RowNum > @MaxRowSelect
```

修改函数后，使用两个用来定义要查看的行范围（这里是3到5）的新参数来调用函数：

```
SELECT RowNum, Val
FROM udf_ParseArray('A,B,C,D,E,F,G', ',', 3, 5)
```

这个查询返回了传给UDF的、字符串数组中的第三、第四和第五个字符。

11.1.5　查看 UDF 元数据

在这个技巧中，会演示如何查看当前数据库中的UDF列表（因为这个查询包含了实际的UDF T-SQL定义，我就不展示结果了）：

```
SELECT name, type_desc, definition
FROM sys.sql_modules s
INNER JOIN sys.objects o
   ON s.object_id = o.object_id
WHERE TYPE IN ('IF', -- 内联表UDF
   'TF', --多语句表UDF
   'FN') --标量UDF
```

解析

sys.sql_modules和sys.objects系统视图用来在查询的结果集中返回UDF名、类型描述以及SQL定义：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FROM sys.sql_modules s
INNER JOIN sys.objects o
   ON s.object_id = o.object_id
```

因为sys.sql_modules包含了其他对象类型的行，所以sys.objects必须限制只返回UDF的行：

```
WHERE TYPE IN('IF', -- 内联表UDF
'TF', --多语句表UDF
'FN') --标量UDF
```

11.1.6 移除用户定义函数

在这个技巧中，我会演示如何删除一个用户定义函数。和其他DROP命令差不多，语法非常简单：

```
DROP FUNCTION { [ schema_name. ] function_name } [ ,...n ]
```

表11-4详细描述了此命令的参数。

356

表11-4 DROP FUNCTION参数

参　数	描　述
[schema_name.] function_name	这个参数定义了可选的架构名和必需的用户定义函数的函数名
[,...n]	尽管不是实际的参数，这个语法参数表示能在一个语句中删除一个或多个用户定义函数

这个技巧演示如何删除前面技巧中创建的dbo.udf_ParseArray函数：

```
DROP FUNCTION dbo.udf_ParseArray
```

解析

尽管有3种不同类型的用户定义函数（标量、内联和多语句），但是只需要使用一个DROP FUNCTION命令就可以删除它们。

你也可以在一个语句中删除多个UDF，例如：

```
DROP FUNCTION dbo.udf_ParseArray, dbo.udf_ReturnEmployeeAddress,
dbo.udf_CheckForSQLInjection
```

11.2 从 UDF 获益

用户定义函数由于进行了执行计划的缓存而能提升性能，并且还能封装可重用代码。在下一部分，将讨论UDF能带来哪些好处。例如，特别是标量函数能让代码更好理解，并且让我们始终能在应用程序中应用查找规则，而不是在不同的存储过程或视图中重复多次相同的代码。

表值函数也很有用，因为使用它可以在结果集上应用参数。例如，使用参数来为数据集定义行级别的安全（稍后会演示）。

警告 创建用户定义函数时需要考虑乘数效应。例如，如果你创建了一个标量用户定义函数来对一个一百万行的表执行查找操作并返回一个值，即使带有索引每一个查找操作也需要30秒，如果你使用这个UDF根据另外一个大表的行返回值，则性能很可能会明显下降。如果标量用户定义函数引用了其他表，请确保你使用的访问表信息的查询性能不错，并且不会返回太大的结果集。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下面的几个技巧会演示用户定义函数的一些比较常见和好的应用。

11.2.1　维护可重用的代码

标量UDF可以把逻辑封装到一个函数中，而不是在每次需要的时候多次重复逻辑，从而减少代码
357 膨胀。

例如，下面的标量用户定义函数用于决定雇员可以拿到的电脑类型。有好几行代码来计算不同的输入参数，包括雇员的头衔、雇员的雇用日期以及是否领薪水。你可以将这些逻辑封装到一个函数中，而不是在数据库中的多个地方重复这段逻辑：

```
CREATE FUNCTION dbo.udf_GET_AssignedEquipment
 (@Title nvarchar(50), @HireDate datetime, @SalariedFlag bit)
RETURNS nvarchar(50)
AS
BEGIN

DECLARE @EquipmentType nvarchar(50)

IF @Title LIKE 'Chief%' OR
   @Title LIKE 'Vice%' OR
   @Title = 'Database Administrator'
BEGIN
  SET @EquipmentType = 'PC Build A'
END

IF @EquipmentType IS NULL AND @SalariedFlag = 1
BEGIN
  SET @EquipmentType = 'PC Build B'
END

IF @EquipmentType IS NULL AND @HireDate < '1/1/2002'
BEGIN
  SET @EquipmentType = 'PC Build C'
END

IF @EquipmentType IS NULL
BEGIN
  SET @EquipmentType = 'PC Build D'
END

RETURN @EquipmentType
END

GO
```

一旦创建了它，就可以在T-SQL代码中的很多地方使用这个标量函数，而不需要重新编码这些逻辑。例如，在查询的SELECT、GROUP BY和ORDER BY子句中使用新的标量函数：

```
SELECT dbo.udf_GET_AssignedEquipment
            (JobTitle, HireDate, SalariedFlag) PC_Build,
       COUNT(*) Employee_Count
FROM HumanResources.Employee
GROUP BY dbo.udf_GET_AssignedEquipment
            (JobTitle, HireDate, SalariedFlag)
ORDER BY dbo.udf_GET_AssignedEquipment
            (JobTitle, HireDate, SalariedFlag)
```

358 这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
PC_Build            Employee_Count
PC Build A          7
PC Build B          45
PC Build C          238
```

第二个查询在SELECT和WHERE子句中使用这个标量函数：

```
SELECT JobTitle,
       BusinessEntityID,
       dbo.udf_GET_AssignedEquipment
            (JobTitle, HireDate, SalariedFlag) PC_Build
FROM HumanResources.Employee
WHERE dbo.udf_GET_AssignedEquipment
           (JobTitle, HireDate, SalariedFlag) IN
                ('PC Build A', 'PC Build B')
```

这个查询返回下面的（部分）结果：

```
JobTitle                        BusinessEntityID    PC_Build
Chief Executive Officer         1                   PC Build A
Vice President of Engineering   2                   PC Build A
Engineering Manager             3                   PC Build B
Design Engineer                 5                   PC Build B
Design Engineer                 6                   PC Build B
...
```

解析

标量用户定义函数能帮助封装业务逻辑，这样就不需要在代码间重复，提供了集中的地方，使得要修改的时候只需要对一个函数修改就可以了。这还提供了一致性，你和其他数据库开发者能一直以相同方式使用和编写相同的逻辑。此外，还有一个优势就是代码可读性，特别是对于执行多个查找或计算的大型查询。

11.2.2 交叉引用的自然键值

回顾第1章可以知道，代理键是人造主键，和自然键（natural key）相反，自然键表示数据的唯一描述符（例如，社会保险号是自然键的例子，而有IDENTITY属性的列就是代理键）。IDENTITY值通常用于代理主键，也被作为外键引用。

在我自己的OLTP（联机事务处理系统）和星型架构数据库设计中，我为每个表默认分配一个代理键，除非有重要的理由才不去这么做。因为这样做能帮我们从外部遗留的自然键抽象出自己的唯一键。例如，如果你使用从HR系统中获取的EmployeeNumber作为主键，当HR系统打算修改数据类型的时候你就会遇到麻烦（强制你修改主键、外键引用以及组合主键）。代理键帮助我们避免诸如此类的修改，因为它们在我们的控制之下，所以它们是非常好的主键。你可以保留自然键的唯一约束而不用担心外部的修改影响你的主键或外键。

把数据从遗留系统导入正式表的时候，常常会需要引用自然键以检测哪些行需要插入、更新或删除。如果只是处理一列（例如EmployeeID、CreditCardNumber、SSN、UPC）就会很简单。然而，如果自然键由多列构成，此时对正式表的交叉引用就不是这么简单了。

下面演示了标量用户定义函数能通过在执行操作前就检查它们的存在性，来简化自然键的查找。要创建这个例子，首先执行一些对象和命令。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

首先，创建一个具有自己的代理键和组成组合自然键的3个列（这3个列形成了从遗留系统获取的唯一值）的表：

```
CREATE TABLE dbo.DimProductSalesperson
(DimProductSalespersonID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
 ProductCD char(10) NOT NULL,
 CompanyNBR int NOT NULL,
 SalespersonNBR int NOT NULL
)
GO
```

然后，创建一个过渡表来保存外部遗留数据文件中的行。例如这个表可以从遗留系统导出的外部文本文件填充。这个表没有主键，因为它只是用来保存要转移到dbo.DimProductSalesperson表中的数据（在转移之前保存）：

```
CREATE TABLE dbo.Staging_PRODSLSP
( ProductCD char(10) NOT NULL,
 CompanyNBR int NOT NULL,
 SalespersonNBR int NOT NULL
)
GO
```

警告　这个技巧中没有为表添加索引，但是在真实环境下，你为了在查询的WHERE子句中使用联结操作或验证，就需要为键列添加索引。

接着，把下面两行插入到过渡表：

```
INSERT dbo.Staging_PRODSLSP
(ProductCD, CompanyNBR, SalespersonNBR)
VALUES ('2391A23904', 1, 24)

INSERT dbo.Staging_PRODSLSP
(ProductCD, CompanyNBR, SalespersonNBR)
VALUES ('X129483203', 1, 34)
```

现在，使用下面的查询将这两行插入到DimProductSalesperson表，其没有使用标量UDF：

```
INSERT dbo.DimProductSalesperson
(ProductCD, CompanyNBR, SalespersonNBR)
SELECT s.ProductCD, s.CompanyNBR, s.SalespersonNBR
FROM dbo.Staging_PRODSLSP s
LEFT OUTER JOIN dbo.DimProductSalesperson d ON
   s.ProductCD = d.ProductCD AND
   s.CompanyNBR = d.CompanyNBR AND
   s.SalespersonNBR = d.SalespersonNBR
WHERE d.DimProductSalespersonID IS NULL
```

360

由于每个列都是自然键，我们必须左联结插入的表和过渡表的每一列，然后使用IS NULL检测行是否已经在目标表中。

还有一种方法能减少每一个INSERT/UPDATE/DELETE中的代码，即使用如下的标量UDF：

```
CREATE FUNCTION dbo.udf_GET_Check_NK_DimProductSalesperson
 (@ProductCD char(10), @CompanyNBR int, @SalespersonNBR int )
RETURNS bit
AS
BEGIN
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DECLARE @Exists bit

IF EXISTS (SELECT DimProductSalespersonID
           FROM dbo.DimProductSalesperson
           WHERE @ProductCD = @ProductCD AND
                 @CompanyNBR = @CompanyNBR AND
                 @SalespersonNBR = @SalespersonNBR)
BEGIN
   SET @Exists = 1
END
ELSE
BEGIN
   SET @Exists = 0
END

RETURN @Exists
END

GO
```

UDF看上去的确代码更多了，但是你之后会在执行数据导入操作的时候获益。例如，现在你可以像下面一样重写前面演示的INSERT操作：

```
INSERT dbo.DimProductSalesperson
(ProductCD, CompanyNBR, SalespersonNBR)
SELECT ProductCD, CompanyNBR, SalespersonNBR
FROM dbo.Staging_PRODSLSP
WHERE dbo.udf_GET_Check_NK_DimProductSalesperson
(ProductCD, CompanyNBR, SalespersonNBR) = 0
```

解析

在这个技巧中，演示了如何创建标量UDF来根据3个参数返回bit值。如果3个值都在正式表的某一行中存在，则返回1，否则返回0。使用这个函数能简化自然键跨多列情况下的INSERT/UPDATE/DELETE代码。

一步步地来看UDF代码，开始的一些行定义了UDF的名字和参数。每个参数都是过渡表和正式表中的组合自然键：

361


```
CREATE FUNCTION dbo.udf_GET_Check_NK_DimProductSalesperson
 (@ProductCD char(10), @CompanyNBR int, @SalespersonNBR int )
```

然后，定义函数要返回的bit数据类型：

```
RETURNS bit
AS
BEGIN
```

创建一个局部变量来保存bit值：

```
DECLARE @Exists bit
```

使用IF来检测匹配所有3个参数到组合自然键的行是否存在。如果存在，则把局部变量设置为1，否则设置为0：

```
IF EXISTS (SELECT DimProductSalespersonID
           FROM dbo.DimProductSalesperson
           WHERE @ProductCD = @ProductCD AND
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                @CompanyNBR = @CompanyNBR AND
                @SalespersonNBR = @SalespersonNBR)
BEGIN
   SET @Exists = 1
END
ELSE
BEGIN
   SET @Exists = 0
END
```

然后把局部变量返回给调用者：

```
RETURN @Exists
END

GO
```

然后在WHERE子句中使用这个函数，从过渡表获取由标量UDF返回0的那些行（它们在DimProduct-Salesperson表中不存在）：

```
WHERE dbo.udf_GET_Check_NK_DimProductSalesperson
(ProductCD, CompanyNBR, SalespersonNBR) = 0
```

11.2.3　使用多语句 UDF 替代视图

多语句UDF允许以和视图一样的方式返回数据，并以和存储过程相似的方式操纵数据。

在本例中，创建一个多语句UDF来根据函数的调用者应用基于行的安全策略。只有某个销售人员相关的行才会被返回。除此之外，第二个参数是一个bit标志位，它控制是否在结果中返回SalesPersonQuotaHistory表中的行：

```
CREATE FUNCTION dbo.udf_SEL_SalesQuota
   ( @BusinessEntityID int,
     @ShowHistory bit )
RETURNS @SalesQuota TABLE
    (BusinessEntityID int,
     QuotaDate datetime,
     SalesQuota money)
AS
BEGIN
    INSERT @SalesQuota
    (BusinessEntityID, QuotaDate, SalesQuota)
    SELECT BusinessEntityID, ModifiedDate, SalesQuota
    FROM Sales.SalesPerson
    WHERE BusinessEntityID = @BusinessEntityID

    IF @ShowHistory = 1
    BEGIN

       INSERT @SalesQuota
       (BusinessEntityID, QuotaDate, SalesQuota)
       SELECT BusinessEntityID, QuotaDate, SalesQuota
       FROM Sales.SalesPersonQuotaHistory
       WHERE BusinessEntityID = @BusinessEntityID

    END

   RETURN
END
```

362

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
GO
```

创建UDF后，执行下面的查询来显示SalesPerson表中某个销售人员的销售指标数据：

```
SELECT BusinessEntityID, QuotaDate, SalesQuota
FROM dbo.udf_SEL_SalesQuota (275,0)
```

这个查询返回：

```
BusinessEntityID    QuotaDate                SalesQuota
275                 2001-06-24 00:00:00.000  300000.00
```

接着，把第二个参数从0换成1，这样就会显示SalespersonID为275的销售人员在SalesPerson-QuotaHistory表中其他一些行：

```
SELECT BusinessEntityID, QuotaDate, SalesQuota
FROM dbo.udf_SEL_SalesQuota (275,1)
```

这个查询返回如下的（部分）结果：

```
BusinessEntityID    QuotaDate                SalesQuota
275                 2001-06-24 00:00:00.000  300000.00
275                 2001-07-01 00:00:00.000  367000.00
275                 2001-10-01 00:00:00.000  556000.00
275                 2002-01-01 00:00:00.000  502000.00
275                 2002-04-01 00:00:00.000  550000.00
275                 2002-07-01 00:00:00.000  1429000.00
275                 2002-10-01 00:00:00.000  1324000.00
...
```

363

解析

这个技巧演示了使用多语句表值UDF来根据传入的BusinessEntityID值返回销售指标数据。它还包含了另一个bit标志位来控制是否返回历史。

看代码后，你会注意到开始的一些行是定义输入参数（视图不允许的操作）：

```
CREATE FUNCTION dbo.udf_SEL_SalesQuota
   ( @BusinessEntityID int,
     @ShowHistory bit )
```

在此之后，定义需要这个函数返回的表列：

```
RETURNS @SalesQuota TABLE
    (BusinessEntityID int,
     QuotaDate datetime,
     SalesQuota money)
```

函数体包括两个独立的批处理语句，第一个是将某个销售人员所在的行插入表变量中的INSERT语句：

```
AS
BEGIN
    INSERT @SalesQuota
    (BusinessEntityID, QuotaDate, SalesQuota)
    SELECT BusinessEntityID, ModifiedDate, SalesQuota
    FROM Sales.SalesPerson
    WHERE BusinessEntityID = @BusinessEntityID
```

然后，IF语句（又一个视图不允许的结构）计算bit参数。如果等于1，则销售指标历史数据会插入到表变量中：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
IF @ShowHistory = 1
    BEGIN

      INSERT @SalesQuota
      (BusinessEntityID, QuotaDate, SalesQuota)
      SELECT BusinessEntityID, QuotaDate, SalesQuota
      FROM Sales.SalesPersonQuotaHistory
      WHERE BusinessEntityID = @BusinessEntityID

END
```

最后，RETURN关键字表示了函数的结束（和标量函数不同，后面没有指定局部变量）：

```
   RETURN
END

GO
```

尽管UDF有视图不允许的T-SQL，但它仍然可以在FROM子句中引用：

```
SELECT BusinessEntityID, QuotaDate, SalesQuota
FROM dbo.udf_SEL_SalesQuota (275,0)
```

虽然结果可以通过在视图中使用UNION语句来返回，但是你不能控制逻辑在一个视图中显示或不显示历史。

364

在本技巧中，演示了如何创建你自己的基于参数的结果集。它可用来实现基于行的安全策略。行级别的安全策略没有在SQL Server安全模型中内建。可以使用函数并通过指定过滤数据的输入参数来只返回那些允许查看的行。

11.3　UDT 基础知识

用户定义类型用于定义一种和已知业务或以应用程序为中心的属性一致的数据类型，比如PIN、PhoneNBR、或EmailAddress。一旦在数据库中创建用户定义类型，就能用于列、参数和变量的定义，提供和底层数据类型的一致性。下面两个技巧会介绍如何创建和删除用户定义类型。注意，和其他的一些数据库对象不同，不能使用ALTER命令修改既有类型。

11.3.1　创建和使用用户定义类型

这个技巧演示了如何创建用户定义类型（也叫做*别名数据类型*），它是设定了用户指定的名字、数据类型、长度和可空性的数据类型的具体配置。你可以使用除新的xml数据类型之外的所有基本数据类型。

警告　使用用户定义数据类型的一个缺点在于它们不能级联修改，在本章的最后一个技巧你会看到。

创建用户定义类型的基本语法如下：

```
CREATE TYPE [ schema_name. ] type_name
{

    FROM base_type
  [ (precision [ ,scale ] ) ]
  [ NULL | NOT NULL ] }
```

表11-5详细描述了这些命令的参数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表11-5 CREATE TYPE参数

参 数	描 述
[schema_name.] type_name	这个参数指定可选的架构名和必需的新的用户定义类型名
base_type	用于定义新的用户定义类型的基础类型。可以使用除xml数据类型以外的所有基本数据类型
(precision [,scale])	如果使用数值基础类型，precision代表存储在小数点左边和右边的最大数字位数。scale代表存储在小数点右边的最大数字位数
NULL \| NOT NULL	这个参数定义新的用户定义类型是否允许NULL值

365

注解 本章讲解了如何使用T-SQL创建用户定义类型。第13章简要讨论了如何使用新的公共语言运行时（CLR）功能创建这些类型。

在本技巧中，我将创建基于14个字符的字符串的新类型：

```
-- 在本例中，假设公司账号在多个表中使用，并且它总是14个字符长度且永远不允许NULL值

CREATE TYPE dbo.AccountNBR
FROM char(14) NOT NULL
GO
```

然后，在两个表的列定义中使用这个新类型：

```
-- 现在在两个不同的表中使用这个新数据类型

CREATE TABLE dbo.InventoryAccount
   (InventoryAccountID int NOT NULL,
   InventoryID int NOT NULL,
   InventoryAccountNBR AccountNBR)
GO

CREATE TABLE dbo.CustomerAccount
   (CustomerAccountID int NOT NULL,
   CustomerID int NOT NULL,
   CustomerAccountNBR AccountNBR)
GO
```

这个类型同样能用于局部变量或输入参数的定义中。例如，下面的存储过程使用新的数据类型来为存储过程定义输入参数：

```
CREATE PROCEDURE dbo.usp_SEL_CustomerAccount
   @CustomerAccountNBR AccountNBR
AS

SELECT CustomerAccountID, CustomerID, CustomerAccountNBR
FROM dbo.CustomerAccount
WHERE CustomerAccountNBR = CustomerAccountNBR
GO
```

然后，使用这个新数据类型创建一个局部变量，并将其传入存储过程：

```
DECLARE @CustomerAccountNBR AccountNBR
SET @CustomerAccountNBR = '1294839482'

EXEC dbo.usp_SEL_CustomerAccount @CustomerAccountNBR
```

要查看用户定义类型的底层基本类型，可以使用sp_help系统存储过程：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
EXEC sp_help 'dbo.AccountNBR'
```

366 这个查询返回（为了演示目的，只显示几列）：

```
Type_name          Storage_type      Length      Nullable
AccountNbr         char              14          no
```

解析

在这个技巧中，以char(14)数据类型和NOT NULL创建了一个叫dbo.AccountNBR的用户定义类型。一旦创建了用户定义类型，就在两个不同的表的列定义中使用它：

```
CREATE TABLE dbo.InventoryAccount
   (InventoryAccountID int NOT NULL,
   InventoryID int NOT NULL,
   InventoryAccountNBR AccountNBR)
```

因为NOT NULL已经在数据类型中指定，所以在列定义中不需要显式定义它。

在创建表之后，我们创建了一个存储过程，它在输入参数定义中使用了这个新的数据类型。然后通过一个使用新类型的局部变量调用存储过程。

尽管T-SQL类型可能对一些开发者来说很方便，不过创建应用程序的数据字典和遵守那些数据类型对他们来说也很有用。例如，如果AccountNBR总是14个字符，DBA或开发者可以沟通和检测以确保新的对象使用一致的名字和数据类型。

11.3.2　找出使用用户定义类型的列和参数

在介绍如何移除用户定义数据类型之前，需要知道如何找出依赖某个类型的所有数据库对象。后面你会看到，移除UDT不会自动级联修改依赖的表。

本例介绍了如何找出哪些数据库对象使用某个用户定义类型。技巧中的第一个查询显示了使用AccountNBR用户定义类型的所有列：

```
SELECT OBJECT_NAME(c.object_id) Table_Name, c.name Column_Name
FROM sys.columns c
INNER JOIN sys.types t ON
   c.user_type_id = t.user_type_id
WHERE t.name = 'AccountNBR'
```

这个查询返回：

```
Table_Name                    Column_Name
InventoryAccount              InventoryAccountNBR
CustomerAccount               CustomerAccountNBR
```

下面的查询显示了具有使用AccountNBR用户定义类型定义的参数的所有过程或函数：

```
-- 现在来看哪些参数引用了AccountNBR数据类型
SELECT OBJECT_NAME(p.object_id) Table_Name, p.name Parameter_Name
FROM sys.parameters p
INNER JOIN sys.types t ON
   p.user_type_id = t.user_type_id
WHERE t.name = 'AccountNBR'
```

367 这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Table_Name                    Parameter_Name
usp_SEL_CustomerAccount       @CustomerAccountNBR
```

解析

要知道哪些表的列使用了用户定义类型，可以使用sys.columns和sys.types系统目录视图：

```
FROM sys.columns c
INNER JOIN sys.types t ON
   c.user_type_id = t.user_type_id
```

sys.columns视图将数据库中的表值函数、表和视图定义的每一列作为一行。sys.types视图将每一个用户或者系统数据类型作为一行。

要找到哪个函数或者存储过程的参数引用了用户定义类型，可以使用sys.parameters和sys.types系统目录视图：

```
FROM sys.parameters p
INNER JOIN sys.types t ON
   p.user_type_id = t.user_type_id
```

sys.parameters视图包含了每一个能接受参数的数据库对象作为一行，例如存储过程。

如果你计划删除用户定义类型，那么找到哪些对象引用了用户定义类型就是必要的，下个技巧会演示。

11.3.3 删除用户定义类型

在这个技巧中，会演示如何从数据库中移除一个用户定义类型（也叫做别名数据类型）。和大多数的DROP命令一样，移除用户定义类型的语法非常简单：

```
DROP TYPE [ schema_name. ] type_name
```

DROP TYPE命令使用架构名和类型名，本技巧将会演示。首先，需要先移除所有对用户定义类型的引用。在本例中，两个表和一个存储过程中的AccountNBR类型被修改为等价的基本类型：

```
ALTER TABLE dbo.InventoryAccount
ALTER COLUMN InventoryAccountNBR char(14)
GO

ALTER TABLE dbo.CustomerAccount
ALTER COLUMN CustomerAccountNBR char(14)
GO

ALTER PROCEDURE dbo.usp_SEL_CustomerAccount
   @CustomerAccountNBR char(14)
AS

SELECT CustomerAccountID, CustomerID, CustomerAccountNBR
FROM dbo.CustomerAccount
WHERE CustomerAccountNBR = CustomerAccountNBR
GO
```

引用的对象现在已经转换了，可以删除这个类型了：

```
DROP TYPE dbo.AccountNBR
```

解析

要移除一个类型，必须先修改或移除数据库表中所有对这个类型的引用。如果要修改UDT的定义，就需要移除所有使用该UDT的数据库对象中对它的所有引用。这意味着删除类型前需要先修改表、视

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

图、存储过程等。如果数据库对象非常依赖它们，那么这个工作非常可怕。而且，如果任何架构绑定的存储过程、函数或触发器使用这个数据类型作为参数或者变量，这些引用必须修改或移除。在这个技巧中，使用ALTER TABLE...ALTER COLUMN把这个数据类型修改为系统数据类型：

```
ALTER TABLE dbo.InventoryAccount
ALTER COLUMN InventoryAccountNBR char(14)
```

存储过程参数也需要使用ALTER PROCEDURE修改：

```
ALTER PROCEDURE usp_SEL_CustomerAccount
(@CustomerAccountNBR char(14))
...
```

11.3.4　传入表值参数

SQL Server 2008引入了可以用来将行集传入到存储过程和用户定义函数的表值参数。这个功能可以使存储过程和函数具有封装多个行集的能力，而不是必须一行一行地调用数据修改过程或创建多个输入参数来生硬地转化为多行。

例如，如下的存储过程使用了几个输入参数向Department表中插入行：

```
CREATE PROCEDURE dbo.usp_INS_Department_Oldstyle
   @Name_1 nvarchar(50),
   @GroupName_1 nvarchar(50),
   @Name_2 nvarchar(50),
   @GroupName_2 nvarchar(50),
   @Name_3 nvarchar(50),
   @GroupName_3 nvarchar(50),
   @Name_4 nvarchar(50),
   @GroupName_4 nvarchar(50),
   @Name_5 nvarchar(50),
   @GroupName_5 nvarchar(50)
AS

INSERT HumanResources.Department
(Name, GroupName)
VALUES (@Name_1, @GroupName_1)

INSERT HumanResources.Department
(Name, GroupName)
VALUES (@Name_2, @GroupName_2)

INSERT HumanResources.Department
(Name, GroupName)
VALUES (@Name_3, @GroupName_3)

INSERT HumanResources.Department
(Name, GroupName)
VALUES (@Name_4, @GroupName_4)

INSERT HumanResources.Department
(Name, GroupName)
VALUES (@Name_5, @GroupName_5)

GO
```

369

前面的示例过程有一些限制。首先，假定每个调用包含5行。如果有10行，就必须调用存储过程两次。如果有3行，就需要修改过程来测试参数中的NULL值并且忽略相应的插入。如果底层的表允许

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使用NULL值，也需要一个方法去指出什么时候应该保存NULL，什么时候不需要保存NULL。

下面是创建一个实现插入的存储过程更常见的技巧：

```
CREATE PROCEDURE dbo.usp_INS_Department_Oldstyle_V2
   @Name nvarchar(50),
   @GroupName nvarchar(50)
AS

INSERT HumanResources.Department
(Name, GroupName)
VALUES (@Name, @GroupName)

GO
```

如果你有5行要插入，就需要调用这个存储过程5次。这在大多数情况下是可以接受的——不过，如果你经常要在一个批处理中插入多行，SQL Server 2008提供了更好的办法。不需要执行一个调用，可以将要插入的值传入到表示值表的一个参数中。这个参数被称为表值参数。

要使用表值参数，第一步要定义用户定义表数据类型，如下面演示的：

```
CREATE TYPE Department_TT AS TABLE
(Name nvarchar(50),
 GroupName nvarchar(50))
GO
```

在数据库中创建好这个新的表类型后，就可以在模块定义以及代码中引用它了：

```
CREATE PROCEDURE dbo.usp_INS_Department_NewStyle
   @DepartmentTable as Department_TT READONLY
AS

INSERT HumanResources.Department
(Name, GroupName)

SELECT Name, GroupName
FROM @DepartmentTable

GO
```

370

假定一个处部进程来填充值列表，值列表是要传入到存储过程中的。在应用程序中，传入的数据源可以从添入的过渡表中生成，从应用程序行集中直接生成，也可以从构造的行集中生成，如下面演示的：

```
-- 我可以声明所有在T-SQL批处理中用到的新类型。
DECLARE @StagingDepartmentTable as Department_TT

-- 插入多行到表类型变量
INSERT @StagingDepartmentTable
(Name, GroupName)
VALUES ('Archivists', 'Accounting')

INSERT @StagingDepartmentTable
(Name, GroupName)
VALUES ('Public Media', 'Legal')

INSERT @StagingDepartmentTable
(Name, GroupName)
VALUES ('Internal Admin', 'Office Administration')

--一次一个地调用将这个表类型变量传入到存储过程。
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
EXEC dbo.usp_INS_Department_NewStyle @StagingDepartmentTable
```

解析

为了将结果集传入到模块，必须先在数据库中定义用户定义表类型。使用CREATE TYPE命令将它定义为AS TABLE：

```
CREATE TYPE Department_TT AS TABLE
```

下一步，定义组成表的两个列，和常规表一样：

```
(Name nvarchar(50),
 GroupName nvarchar(50))
GO
```

我还用PRIMARY KEY、UNIQUE和CHECK约束定义了表类型。也可以指定是否为空以及定义是否为计算列。

接着，创建使用新创建的表类型的存储过程。在输入参数变量列表中，使用Department_TT类型创建输入参数：

```
CREATE PROCEDURE dbo.usp_INS_Department_NewStyle
   @DepartmentTable as Department_TT READONLY
AS
```

注意在数据类型指定之后的READONLY关键字。这是存储过程和用户定义函数输入参数需要的，因为在SQL Server这个版本中不允许更改表值结果集。

371 下面的代码块是在表上执行INSERT操作，其使用输入参数作为多行的数据源：

```
INSERT HumanResources.Department
(Name, GroupName)
SELECT Name, GroupName
FROM @DepartmentTable

GO
```

在这之后，演示了声明将传入到存储过程的包含多行的局部变量。DECLARE语句定义了变量的名称，随后是在技巧的前面定义好的用户定义类型表的名称：

```
DECLARE @StagingDepartmentTable as Department_TT
```

声明之后，将多行插入到表中，然后将它作为存储过程调用的参数传入：

```
INSERT @StagingDepartmentTable
(Name, GroupName)
VALUES ('Archivists', 'Accounting')
...

EXEC dbo.usp_INS_Department_NewStyle @StagingDepartmentTable
```

在处理业务进程的存储过程时这个新功能会很有用。例如，如果你有一个处理产品订单的网站，现在可以传入结果集到包含普通头信息的一个存储过程中，还有表示被定购产品的多行。这个应用程序进程可以构造为一个调用，而不是为每个唯一的定购的产品线物品进行多次调用。对于特别忙的系统，使用表值参数减少了应用程序与数据库服务器之间的交互，从而增加了网络带宽且在SQL Server
372 端的事务批处理也更高效。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第12章 触 发 器

在本章中，会介绍一些技巧来创建和使用数据操作语言（DML）和数据定义语言（DDL）触发器。DML触发器（trigger）包含用于对表或视图的INSERT、UPDATE或DELETE操作做出响应的T-SQL代码。DDL触发器对服务器或数据库事件作出响应而不是数据修改。例如，你可以创建一个在数据库用户发出CREATE TABLE或DROP TABLE命令的时候写入审核表的DDL触发器。

提示 在SQL Server 2008中，执行DDL操作的系统存储过程现在会触发DDL触发器。

触发器在正确使用时能够很方便且自动地响应某种行为。对于必须对某种行为做出业务级别响应的情况来说触发器很合适。触发器不应该替代约束（例如主键或唯一性约束），因为约束更适合这些操作并且执行得更好。你同样应该了解用于定义触发器的T-SQL，确保代码被适当优化。如果对于每个UPDATE的触发器需要几秒钟的时间来执行，那么你会遇到整体的性能问题。

按照我的经验，检查性能问题时触发器总是容易被人忽略。我会听到抱怨说执行数据修改的性能非常差，并花了很多时间尝试优化，最后发现是没有调优的触发器引起的性能问题。这也就是我很少使用DML触发器的主要原因之一——当确实要用的时候，我会特别小心来确保它们是快速的且没有bug。然而，应用程序可能需要使用DML触发器。更不要说SQL Server DDL触发器开启了之前版本未提供的全新功能，提供了一些其他数据库对象类型不能轻易替代的特性。

在本章中，我会介绍下面的主题：

- 如何创建AFTER DML触发器；
- 如何创建INSTEAD OF DML触发器；
- 如何创建DDL触发器；
- 如何修改或删除既有触发器；
- 如何启用和禁用触发器；
- 如何限制触发器嵌套、设置触发顺序和控制递归；
- 如何查看触发器元数据；
- 如何使用触发器来响应登录事件。

首先我们来讨论一下DML触发器的背景知识。

注解 本章介绍了如何使用T-SQL创建触发器。并且，第13章将会介绍使用新的公共语言运行时（CLR）功能创建触发器。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

12.1 DML触发器

DML触发器对用户对表或视图的INSERT、UPDATE或DELETE操作作出响应。当数据修改事件发生的时候，触发器执行触发器内定义的一组行为。和存储过程相似，触发器也使用T-SQL定义并允许执行所有操作。

DML触发器能特别定义为FOR UPDATE、FOR INSERT、FOR DELETE或这3者的任何组合。UPDATE触发器能对表中一个或多列的修改作出响应，INSERT触发器对插入数据库的新数据作出响应，而DELETE触发器对将从数据库中删除的数据作出响应。有两种类型的DML触发器：AFTER和INSTEAD OF。

AFTER触发器只对表可用，并且它们在对表的数据修改完成后执行。INSTEAD OF触发器的执行会替代原始的数据修改，并且对表和视图都能创建。

使用DML触发器可以对表中的数据修改作出响应。例如，可以创建一个触发器来根据执行的操作填充审核表或者使用触发器来减少数量值。尽管触发器自动行为非常强大，但在全面使用触发器之前还需要注意下面一些点。

- 触发器通常很隐蔽，因此也就容易忘记。在检查性能或逻辑问题的时候，DBA会忘记触发器在后台执行。确保触发器在你的数据文档中是“可见的”。
- 如果所有的数据修改流程都是通过存储过程完成，我强烈推荐你在存储过程中执行所有活动，而不是使用触发器。例如，如果你需要更新相关表中的数量，在插入销售记录后，为什么不把这个逻辑放在存储过程中呢？这样的优势在于当存储过程需要修改或发生了预期之外的行为时有可管理性（有地方查看）和可维护性（有地方调试）。
- 始终需要保证性能，也就是说触发器需要能快速执行。长时间运行的触发器会严重减慢数据修改操作。在数据修改频繁的数据库中使用触发器需要特别小心。
- 不记录日志的更新不会引起DML触发器的触发（例如WRITETEXT、TRUNCATE TABLE以及批量插入操作）。
- 约束通常比DML触发器运行更快，因此如果约束能满足你的业务需求，则使用约束来替代。AFTER触发器在数据修改发生后进行，因此它们不能防止约束的违反。
- 不允许在触发器中使用SELECT语句返回结果集。大多数应用程序不能以优雅的方式处理这些数据，而且嵌入的查询会影响触发器的性能。

只要记住这些大致的准则，并且正确使用它们，那么触发器就是在数据库中强制一些业务规则的好方法。

374

警告 本章中一部分演示的触发器会与SQL实例和数据库中既有的触发器冲突，如果你尝试这些代码，请一定只在SQL Server的开发环境下测试这些功能。

12.1.1 创建AFTER DML触发器

AFTER DML触发器在对表的INSERT、UPDATE和/或DELETE修改操作成功完成后执行。ALTER DML触发器的具体语法如下：

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON table
[ WITH <dml_trigger_option> [ ...,n ] ]
AFTER
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
[ NOT FOR REPLICATION ]
AS { sql_statement [ ...n ]}
```

表12-1详细描述了这个命令的参数。

表12-1 CREATE TRIGGER参数

参 数	描 述
[schema_name .]trigger_name	定义了可选的架构拥有者以及必需的用户定义的新触发器名
table	定义了应用触发器的表名
<dml_trigger_option> [...,n]	允许指定ENCRYPTION和/或EXECUTE AS子句。ENCRYPTION会加密触发器的T-SQL定义，使之在系统表中不可见。EXECUTE AS允许定义触发器执行的安全上下文
{ INSERT } [,] [UPDATE] [,] [DELETE]	定义哪个或哪些DML事件需要触发器作出响应，包括INSERT、UPDATE和DELETE。一个触发器能对表的一个或多个操作作出响应
NOT FOR REPLICATION	指定在对表进行复制修改的时候不能执行触发器
sql_statement [...n]	允许一个或多个T-SQL语句。它能用于执行诸如对DML修改进行确认或进行其他表DML行为的操作

在介绍技巧之前比较重要的事情就是为触发器在SQL Server中创建两个“虚拟”表，分别叫做deleted和inserted表。这两个表捕获修改行修改前后的样子。表12-2显示了每一个DML操作影响的表。 375

表12-2 Inserted和Deleted虚拟表

DML操作	Inserted表存放	Deleted表存放
INSERT	插入的行	
UPDATE	新行（更新后的行）	旧行（更新前的行）
DELETE		删除的行

inserted和deleted表能在触发器中使用，用来访问引起触发器触发的数据修改之前或之后的数据。这些表会存储一行或多行修改的数据。确保使用两种类型的更新（一行或多行）编写触发器。例如，一个DELETE操作能影响一行或者50行——所以请确保触发器能进行相应的处理。

在这个技巧中，我会演示使用触发器来跟踪Production.ProductInventory表中行的插入和删除：

```
USE AdventureWorks
GO

-- 跟踪所有的插入、更新和删除
CREATE TABLE Production.ProductInventoryAudit
   (ProductID int NOT NULL ,
    LocationID smallint NOT NULL ,
    Shelf nvarchar(10) NOT NULL ,
    Bin tinyint NOT NULL ,
    Quantity smallint NOT NULL ,
    rowguid uniqueidentifier NOT NULL ,
    ModifiedDate datetime NOT NULL ,
    InsOrUPD char(1) NOT NULL )
GO

-- 创建触发器来填充Production.ProductInventoryAudit表
CREATE TRIGGER Production.trg_uid_ProductInventoryAudit
ON Production.ProductInventory
AFTER INSERT, DELETE
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
AS

SET NOCOUNT ON

--插入的行
INSERT Production.ProductInventoryAudit
(ProductID, LocationID, Shelf, Bin, Quantity,
rowguid, ModifiedDate, InsOrUPD)
SELECT DISTINCT i.ProductID, i.LocationID, i.Shelf, i.Bin, i.Quantity,
i.rowguid, GETDATE(), 'I'
FROM inserted i

-- 删除的行

INSERT Production.ProductInventoryAudit
(ProductID, LocationID, Shelf, Bin, Quantity,
rowguid, ModifiedDate, InsOrUPD)
SELECT d.ProductID, d.LocationID, d.Shelf, d.Bin, d.Quantity,
d.rowguid, GETDATE(), 'D'
FROM deleted d
```

376

```
GO

-- 插入一个新行
INSERT Production.ProductInventory
(ProductID, LocationID, Shelf, Bin, Quantity)
VALUES (316, 6, 'A', 4, 22)

-- 删除一行

DELETE Production.ProductInventory
WHERE ProductID = 316 AND
   LocationID = 6

-- 检测审核表
SELECT ProductID, LocationID, InsOrUpd
FROM Production.ProductInventoryAudit
```

这个查询返回：

```
ProductID       LocationID      InsOrUpd
316             6               I
316             6               D
```

解析

这个技巧从新建一个保存Production.ProductInventory表中已插入和已删除行的表开始。新表的架构和原始表匹配，只是这次新加了一个叫InsOrUPD的列来指示该行是INSERT操作还是UPDATE操作：

```
CREATE TABLE Production.ProductInventoryAudit
   (ProductID int NOT NULL ,
     LocationID smallint NOT NULL ,
    Shelf nvarchar(10) NOT NULL ,
    Bin tinyint NOT NULL ,
    Quantity smallint NOT NULL ,
    rowguid uniqueidentifier NOT NULL ,
    ModifiedDate datetime NOT NULL ,
     InsOrUPD char(1) NOT NULL )
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接着，使用CREATE TRIGGER创建一个AFTER DML触发器。在语句的第一行指定了拥有者架构和新的触发器名：

```
CREATE TRIGGER Production.trg_uid_ProductInventoryAudit
```

在ON子句中指定表（这个表更新的时候会触发触发器）：

```
ON Production.ProductInventory
```

两种类型的DML活动会被监视：插入和删除：

```
AFTER INSERT, DELETE
```

377

触发器的主体在AS关键字后开始：

```
AS
```

SET NOCOUNT设置成ON来屏蔽触发器触发时“行受影响”消息返回给调用的应用程序：

```
SET NOCOUNT ON
```

第一行语句向新的审核表插入一个虚拟的inserted表中既有的行：

```
INSERT Production.ProductInventoryAudit
(ProductID, LocationID, Shelf, Bin, Quantity,
rowguid, ModifiedDate, InsOrUPD)
SELECT DISTINCT i.ProductID, i.LocationID, i.Shelf, i.Bin, i.Quantity,
i.rowguid, GETDATE(), 'I'
FROM inserted i
```

第二行语句向新的审核表插入一个deleted虚拟表（而不是inserted表）中既有的行：

```
INSERT Production.ProductInventoryAudit
(ProductID, LocationID, Shelf, Bin, Quantity,
rowguid, ModifiedDate, InsOrUPD)
SELECT d.ProductID, d.LocationID, d.Shelf, d.Bin, d.Quantity,
d.rowguid, GETDATE(), 'D'
FROM deleted d

GO
```

在创建触发器后，向Production.ProductInventory表插入一个新行，然后马上删除它以测试触发器：

```
-- 插入一个新行
INSERT Production.ProductInventory
(ProductID, LocationID, Shelf, Bin, Quantity)
VALUES (316, 6, 'A', 4, 22)

-- 删除一行

DELETE Production.ProductInventory
WHERE ProductID = 316 AND
   LocationID = 6
```

12

之后对审核表进行查询，会看到有两行跟踪了对Production.ProductInventory表的插入和删除操作：

```
SELECT ProductID, LocationID, InsOrUpd
FROM Production.ProductInventoryAudit
```

12.1.2 创建 INSTEAD OF DML 触发器

INSTEAD OF DML触发器的执行替代触发触发器的原始数据修改操作，并且对表和视图都允许。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

INSTEAD OF触发器通常用于处理不允许进行数据修改的视图的数据修改操作（有关视图成为可更新的所需要的规则请参阅第7章）。

DML触发器使用下面的语法：

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ...,n ] ]
INSTEAD OF
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ NOT FOR REPLICATION ]
AS { sql_statement [ ...n ] }
```

378

表12-3详细描述了这个命令的参数。

表12-3　INSTEAD OF触发器参数

参　数	描　述
[schema_name .]trigger_name	定义了可选的架构拥有者以及必需的用户定义的新触发器名
table \| view	定义了应用触发器的表名或视图名
<dml_trigger_option> [...,n]	允许指定ENCRYPTION和/或EXECUTE AS子句。ENCRYPTION会加密触发器的T-SQL定义，使之在系统表中不可见。EXECUTE AS允许定义触发器执行的安全上下文
[INSERT] [,] [UPDATE] [,] [DELETE]	定义哪个或哪些DML事件需要触发器作出响应，包括INSERT、UPDATE和DELETE。一个触发器能对表的一个或多个操作作出响应
NOT FOR REPLICATION	指定在对表进行复制修改的时候不执行触发器
sql_statement [...n]	允许一个或多个T-SQL语句。它能用于执行诸如对DML修改进行确认或进行其他表DML行为的操作

在这个技巧中，我们会创建一个新表来保存HumanResources.Department表“等待批准”（pending approval）行。这些是需要管理者的批准才能加入正式表的新部门。我们会创建一个视图来显示两个表中所有“已批准”和“等待批准”的部门，然后会在视图上创建一个INSTEAD OF触发器，导致插入的行会被转到新的审批表，而不是HumanResources.Department表：

```
USE AdventureWorks
GO

-- 创建部门“审批”表
CREATE TABLE HumanResources.DepartmentApproval
   (Name nvarchar(50) NOT NULL UNIQUE,
     GroupName nvarchar(50) NOT NULL,
     ModifiedDate datetime NOT NULL DEFAULT GETDATE())
GO


-- 创建视图来查看已批准的和待批准的部门
CREATE VIEW HumanResources.vw_Department
AS
SELECT Name, GroupName, ModifiedDate, 'Approved' Status
FROM HumanResources.Department
UNION
SELECT Name, GroupName, ModifiedDate, 'Pending Approval' Status
FROM HumanResources.DepartmentApproval
```

379

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
GO

-- 在新视图上创建INSTEAD OF触发器
CREATE TRIGGER HumanResources.trg_vw_Department
ON HumanResources.vw_Department
INSTEAD OF
INSERT
AS

SET NOCOUNT ON
INSERT HumanResources.DepartmentApproval
(Name, GroupName)
SELECT i.Name, i.GroupName
FROM inserted i
WHERE i.Name NOT IN (SELECT Name FROM HumanResources.DepartmentApproval)

GO


--尽管视图连接了两个表，向新视图插入行
INSERT HumanResources.vw_Department
(Name, GroupName)
VALUES ('Print Production', 'Manufacturing')


-- 检查视图的内容
SELECT Status, Name
FROM HumanResources.vw_Department
WHERE GroupName = 'Manufacturing'
```

这个查询返回下面的结果集：

```
Status              Name
Approved            Production
Approved            Production Control
Pending Approval    Print Production
```

解析

这个技巧首先创建表用于保存“等待批准”的部门行：

```
CREATE TABLE HumanResources.DepartmentApproval
   (Name nvarchar(50) NOT NULL UNIQUE,
    GroupName nvarchar(50) NOT NULL,
    ModifiedDate datetime NOT NULL DEFAULT GETDATE())
```

然后，创建视图来显示“已批准”和“等待批准”的部门：

```
CREATE VIEW HumanResources.vw_Department
AS

SELECT Name, GroupName, ModifiedDate, 'Approved' Status
FROM HumanResources.Department
UNION
SELECT Name, GroupName, ModifiedDate, 'Pending Approval' Status
FROM HumanResources.DepartmentApproval

GO
```

CREATE VIEW中的UNION会导致视图不能更新，因为对它的任何插入都是不明确的。INSTEAD OF触发

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

器可以对不能更新的视图进行数据修改。

创建触发器来对INSERT进行响应，只要部门名是唯一的话就把它们转到审批表：

```
CREATE TRIGGER HumanResources.trg_vw_Department
ON HumanResources.vw_Department
INSTEAD OF
INSERT
AS

SET NOCOUNT ON
INSERT HumanResources.DepartmentApproval
(Name, GroupName)
SELECT i.Name, i.GroupName
FROM inserted i
WHERE i.Name NOT IN (SELECT Name FROM HumanResources.DepartmentApproval)
```

对视图进行一个新的INSERT测试，查看是否会在审批表中插入数据：

```
INSERT HumanResources.vw_Department
(Name, GroupName)
VALUES ('Print Production', 'Manufacturing')
```

然后查询视图，表明行已经被插入并且显示了“等待批准状态”。

12.1.3　在 DML 触发器中处理事务

在这个技巧中，会演示DML触发器的使用和它们与事务的交互——在触发器内和引起触发器触发的初始化事件内。对于这些示例，我们会使用那些在12.1.1节的技巧中创建的对象。

当触发器触发的时候，SQL Server总是会为它创建一个事务，允许由触发器或调用方做出的任何修改回滚到之前的状态。例如，当遇到某个Shelf或Quantity值的时候，重写trg_uid_ProductInventory-Audit触发器会失败。如果是这样的话，可以使用ROLLBACK来取消触发器并撤销任何修改：

```
USE AdventureWorks
GO

IF EXISTS
(SELECT 1
 FROM sys.triggers
 WHERE object_id =
   OBJECT_ID(N'[Production].[trg_uid_ProductInventoryAudit]'))
DROP TRIGGER [Production].[trg_uid_ProductInventoryAudit]
GO

CREATE TRIGGER Production.trg_uid_ProductInventoryAudit
ON Production.ProductInventory
AFTER INSERT, DELETE
AS

SET NOCOUNT ON

IF EXISTS
(SELECT Shelf
FROM inserted
WHERE Shelf = 'A')
BEGIN
   PRINT 'Shelf ''A'' is closed for new inventory.'
   ROLLBACK
END
```

381

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
-- 插入的行
INSERT Production.ProductInventoryAudit
(ProductID, LocationID, Shelf, Bin, Quantity,
rowguid, ModifiedDate, InsOrUPD)
SELECT DISTINCT i.ProductID, i.LocationID, i.Shelf, i.Bin, i.Quantity,
i.rowguid, GETDATE(), 'I'
FROM inserted i

--删除的行

INSERT Production.ProductInventoryAudit
(ProductID, LocationID, Shelf, Bin, Quantity,
rowguid, ModifiedDate, InsOrUPD)
SELECT d.ProductID, d.LocationID, d.Shelf, d.Bin, d.Quantity,
d.rowguid, GETDATE(), 'D'
FROM deleted d

IF EXISTS
(SELECT Quantity
FROM deleted
WHERE Quantity > 0)
BEGIN
   PRINT 'You cannot remove positive quantity rows!'
   ROLLBACK
END

GO
```

现在使用Shelf“A”插入新行来测试：

```
INSERT Production.ProductInventory
(ProductID, LocationID, Shelf, Bin, Quantity)
VALUES (316, 6, 'A', 4, 22)
```

因为按照触发器逻辑这是不允许的，触发器既不会把行插入审核表也不允许调用INSERT： 382

```
Shelf 'A' is closed for new inventory.
消息3609，级别16，状态1，第2行
事务在触发器中结束。批处理已中止。
```

在前面的示例中，引起触发器触发的INSERT没有显式使用事务。但是，操作仍然被回滚。下面的示例演示了两个删除操作，一个允许（根据触发器的规则）而另外一个不允许。两个插入都嵌在显式事务内：

```
BEGIN TRANSACTION

-- 删除0数量的行
DELETE Production.ProductInventory
WHERE ProductID = 853 AND
   LocationID = 7

-- 删除非0数量的行
DELETE Production.ProductInventory
WHERE ProductID = 999 AND
   LocationID = 60

COMMIT TRANSACTION
```

这个查询返回下面的输出：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
(1行受影响)
You cannot remove positive quantity rows!
消息3609，级别16，状态1，第9行
事务在触发器中结束。批处理已中止。
```

因为触发器发起了回滚，所以外部事务也失效了（也就是说不会保持打开）。而且，即使第一行是有效的删除，因为它们都在一个事务中调用，所有没有行被删除：

```
SELECT ProductID, LocationID
FROM Production.ProductInventory
WHERE (ProductID = 853 AND
   LocationID = 7) OR
(ProductID = 999 AND
   LocationID = 60)
```

这个查询返回：

```
ProductID          LocationID
853                7
999                60
```

解析

这个技巧演示了触发器和事务的交互。简单来说，如果触发器发起了ROLLBACK命令，那么任何由触发器进行的数据修改操作或事务中其余的语句都不会完成。调用触发器的T-SQL查询或批处理也会被取消或回滚。如果激活调用方嵌入在显式事务内，则整个调用事务会被取消和回滚。如果在显式事务中使用触发器，SQL Server会把它当作嵌套事务。在第3章中提到过，ROLLBACK会回滚所有事务，不管嵌套了多少层。

383

12.1.4　根据修改的列控制 DML 触发器

当触发器触发的时候，可以通过使用UPDATE函数来检测哪些列修改了。

不要和DML命令混淆，如果在一个列上执行了INSERT或DML UPDATE操作，UPDATE会返回一个TRUE值。例如，下面的DML UPDATE触发器检测某个列是否已经被修改，如果是，则返回一个错误或回滚这些修改：

```
USE AdventureWorks
GO

CREATE TRIGGER HumanResources.trg_U_Department
ON HumanResources.Department
AFTER UPDATE
AS

IF UPDATE(GroupName)
BEGIN
   PRINT 'Updates to GroupName require DBA involvement.'
   ROLLBACK
END
GO
```

在下一个查询中尝试更新GroupName值：

```
UPDATE HumanResources.Department
SET GroupName = 'Research and Development'
WHERE DepartmentID = 10
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回下面的警告消息，这个错误告诉我们批处理已经被中止（没有进行更新）：

```
Updates to GroupName require DBA involvement.
消息3609，级别16，状态1，第1行
事务在触发器中结束。批处理已中止。
```

解析

如果触发器逻辑针对更细粒度的基于列的修改，则使用UPDATE函数和条件处理来确保代码只针对某些列来执行。在条件处理中嵌入逻辑有助于减少每次触发器触发时的开销——至少对于那些和触发器用途不相关的列。 384

12.1.5 查看DML触发器元数据

下面的技巧演示如何查看当前数据库中有关触发器的信息。

下面第一个示例查询sys.triggers目录视图，返回视图名或表名、相关的触发器名、触发器是否为INSTEAD OF和触发器是否禁止了：

```
-- 显示当前数据库中的DML触发器
SELECT OBJECT_NAME(parent_id) Table_or_ViewNM,
       name TriggerNM,
       is_instead_of_trigger,
       is_disabled
FROM sys.triggers
WHERE parent_class_desc = 'OBJECT_OR_COLUMN'
ORDER BY OBJECT_NAME(parent_id), name
```

这个查询返回下面的（部分）结果：

```
Table_or_ViewNM     TriggerNM           is_instead_of_trigger   is_disabled
Department          trg_U_Department    0                       0
Employee            dEmployee           1                       0
Person              iuPerson            0                       0
...
Vendor              dVendor             1                       0
vw_Department       trg_vw_Department   1                       0
WorkOrder           iWorkOrder          0                       0
WorkOrder           uWorkOrder          0                       0
```

要显示某个触发器的T-SQL定义，可以查询sys.sql_modules系统目录视图：

```
-- 显示触发器的SQL定义
-- (如果触发器没有加密)
SELECT o.name, m.definition
FROM sys.sql_modules m
INNER JOIN sys.objects o ON
     m.object_id = o.object_id
WHERE o.type = 'TR'
```

解析

技巧中的第一个查询查询了sys.triggers目录视图来显示当前数据库中的所有DML触发器。在sys.triggers目录视图中也有DDL触发器，所以我们将parent_class_desc限制为OBJECT_OR_COLUMN来防止它们在结果中显示。你会在下面的技巧中看到，这是因为DDL触发器是属于数据库或SQL Server实例范围内的，而不是在架构范围。

第二个查询显示了数据库中每一个T-SQL触发器真实的名字和定义。如果触发器被加密了（和加密的视图或存储过程相似），查询中的触发器定义会显示为NULL值。 385

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

12.2　DDL触发器

DDL触发器是对服务器或数据库事件作出响应，而不是表数据修改。例如，可以对审核表创建一个DDL触发器，只要数据库用户使用CREATE TABLE或DROP TABLE命令都会触发。或者，在服务器级别，可以创建DDL触发器对新建登录名作出响应（例如，阻止新建某个登录名）。

提示　在SQL Server 2008中，执行DDL操作的系统存储过程会触发DDL触发器。例如，sp_create_plan_guide和sp_control_plan_guide将触发CREATE_PLAN_GUIDE事件。

数据库DDL触发器会作为对象保存在创建它们的数据库中，而服务器DDL触发器在服务器级别跟踪修改，它们会保存在master数据库中。

DDL触发器的语法如下：

```
CREATE TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH <ddl_trigger_option> [ ...,n ] ]
FOR { event_type | event_group } [ ,...n ]
AS { sql_statement [ ...n ]}
```

表12-4详细描述了此命令的参数。

表12-4　CREATE TRIGGER (DDL) 参数

参　数	描　述
trigger_name	这个参数是用户定义的新DDL触发器名（注意，DDL触发器没有拥有者架构，因为它不关联到真实的数据库表或视图）
ALL SERVER \| DATABASE	这个参数指定DDL触发器是对服务器范围（ALL SERVER）还是DATABASE范围的事件作出响应
<ddl_trigger_option> [...,n]	这个参数允许指定ENCRYPTION和/或EXECUTE AS子句。ENCRYPTION会加密触发器的T-SQL定义，使之在系统表中不可见。EXECUTE AS允许定义触发器执行的安全上下文
{ event_type \| event_group } [,...n]	event_type指定需要由触发器作出响应的某个DDL服务器或数据库事件。例如，CREATE_TABLE、ALTER_TABLE、DROP_INDEX等。event_group是event_type事件的逻辑分组。可以创建一个DDL触发器来对一个或多个事件类型或组作出响应。例如，DDL_PARTITION_FUNCTION_ EVENTS组对下面各个事件作出响应：CREATE_PARTITION_FUNCTION、ALTER_PARTITION_FUNCTION和DROP_PARTITION_FUNCTION。你可以在SQL Server联机丛书的“用于激发DDL触发器的DDL事件”主题中找到触发器事件类型的完整列表，并在“用于激发DDL触发器的事件组”主题中找到触发器事件组的完整列表
sql_statement [...n]	这个参数定义了能用于对DDL数据库或服务器事件作出响应的一个或多个T-SQL语句

386

12.2.1　创建审核数据库级别的事件的DDL触发器

这个技巧演示创建一个包含在AdventureWorks数据库上创建、修改或删除索引请求的任何信息的审核表。

首先，创建审核表：

```
USE master
GO

CREATE TABLE dbo.ChangeAttempt
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
   (EventData xml NOT NULL,
     AttemptDate datetime NOT NULL DEFAULT GETDATE(),
     DBUser char(50) NOT NULL)
GO
```

接着，创建一个数据库DDL触发器来跟踪索引操作，在新创建的表中插入事件数据：

```
CREATE TRIGGER db_trg_RestrictINDEXChanges
ON DATABASE
FOR CREATE_INDEX, ALTER_INDEX, DROP_INDEX
AS

SET NOCOUNT ON

INSERT dbo.ChangeAttempt
(EventData, DBUser)
VALUES (EVENTDATA(), USER)

GO
```

现在，尝试在数据库中创建真实索引：

```
CREATE NONCLUSTERED INDEX ni_ChangeAttempt_DBUser ON
     dbo.ChangeAttempt(DBUser)
GO
```

接着，会查询ChangeAttempt审核表来看看新索引的创建事件是不是被触发器捕获了：

```
SELECT EventData
FROM dbo.ChangeAttempt
```

它返回了使用XML格式保存的实际事件信息（更多有关SQL Server中XML的信息请参阅第14章）：

```
<EVENT_INSTANCE>
  <EventType>CREATE_INDEX</EventType>
  <PostTime>2008-02-26T11:29:38.480</PostTime>
  <SPID>53</SPID>
  <ServerName>CAESAR\AUGUSTUS</ServerName>
  <LoginName>CAESAR\Administrator</LoginName>
  <UserName>dbo</UserName>
  <DatabaseName>AdventureWorks</DatabaseName>
  <SchemaName>dbo</SchemaName>
  <ObjectName>ni_ChangeAttempt_DBUser</ObjectName>
  <ObjectType>INDEX</ObjectType>
  <TargetObjectName>ChangeAttempt</TargetObjectName>
  <TargetObjectType>TABLE</TargetObjectType>
  <TSQLCommand>
    <SetOptions ANSI_NULLS="ON" ANSI_NULL_DEFAULT="ON"
     ANSI_PADDING="ON" QUOTED_IDENTIFIER="ON" ENCRYPTED="FALSE" />
    <CommandText>CREATE NONCLUSTERED INDEX ni_ChangeAttempt_DBUser ON
     dbo.ChangeAttempt(DBUser)
</CommandText>
  </TSQLCommand>
</EVENT_INSTANCE>
```

解析

这个技巧从创建表开始，它可以包含创建登录和修改索引的审核信息。EventData列使用SQL Server的xml数据类型，它由新的EVENTDATA函数填充（在这个技巧的后面会介绍）：

```
CREATE TABLE dbo.ChangeAttempt
   (EventData xml NOT NULL,
   AttemptDate datetime NOT NULL DEFAULT GETDATE(),
```

387
12
Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    DBUser char(50) NOT NULL)
GO
```

技巧中创建的第一个触发器应用到当前的数据库。新的DLL触发器对CREATE INDEX、ALTER INDEX或DROP INDEX命令作出响应：

```
CREATE TRIGGER db_trg_RestrictINDEXChanges
ON DATABASE
FOR CREATE_INDEX, ALTER_INDEX, DROP_INDEX
AS
```

在触发器中使用SET NOCOUNT命令来禁止SQL Server中行受影响的消息（否则每次你修改索引的时候都会看到“1行受影响”的消息）：

```
SET NOCOUNT ON
```

然后是对新审核表进行INSERT操作，它填充事件数据和用户：

```
INSERT dbo.ChangeAttempt
(EventData, DBUser)
VALUES (EVENTDATA(), USER)

GO
```

EVENTDATA函数以XML格式返回服务器和数据事件信息，它还能用于SQL Server的SQL Service Broker功能。

注解　更多有关事件通知的信息请参阅第20章。

388 由EVENTDATA函数捕获的XML数据包含了诸如事件、试图CREATE INDEX的登录名、目标对象名以及发生时间等有用信息。

12.2.2　创建审核服务器级别的事件的 DDL 触发器

在这个技巧中，会演示使用服务器级别的DDL触发器来限制用户在SQL Server实例上创建新的登录名。

首先创建DDL触发器：

```
USE master
GO

-- 不允许在SQL实例上有新的登录名
CREATE TRIGGER srv_trg_RestrictNewLogins
ON ALL SERVER
FOR CREATE_LOGIN
AS
PRINT 'No login creations without DBA involvement.'

   ROLLBACK
GO
```

接着，试图增加新的SQL登录名：

```
CREATE LOGIN JoeS WITH PASSWORD = 'A235921'
GO
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
No login creations without DBA involvement.
消息3609，级别16，状态2，第1行
事务在触发器中结束。批处理已中止。
```

在12.3.6节的技巧中会讨论DROP TRIGGER命令，下面是删除这个触发器的清空代码：

```
DROP TRIGGER srv_trg_RestrictNewLogins
ON ALL SERVER
```

解析

这个技巧演示了使用服务器级别的DDL触发器来限制创建SQL登录名。触发器的FOR语句被设置为CREATE LOGIN事件：

```
CREATE TRIGGER srv_trg_RestrictNewLogins
ON ALL SERVER
FOR CREATE_LOGIN
AS
```

触发器的主体使用PRINT语句来警告终端用户的尝试不被允许：

```
PRINT 'No login creations without DBA involvement.'
```

它后面为ROLLBACK命令，该命令取消了触发器的CREATE LOGIN尝试：

```
    ROLLBACK
GO
```

389

12.2.3 使用登录触发器

登录触发器同步响应SQL Server实例的登录事件。可以使用登录触发器来创建对特定登录事件的响应或者只是跟踪登录事件的信息。

警告 设计登录触发器时要非常小心。在部署到生产环境前要先在开发环境中进行测试。如果你要使用登录触发器来限制SQL Server实例的入口，小心不要限制了所有的访问。

在这个技巧中，将演示创建登录触发器限制登录名在某个时间段时对SQL Server的访问。我也在另一个表中记录登录尝试的日志信息。

首先，新建登录名：

```
CREATE LOGIN nightworker WITH PASSWORD = 'E74A53C6'
GO
```

下一步，创建审核表和跟踪登录尝试的表：

```
CREATE DATABASE ExampleAuditDB
GO

USE ExampleAuditDB
GO

CREATE TABLE dbo.RestrictedLogonAttempt
   (LoginNM sysname NOT NULL,
    AttemptDT datetime NOT NULL)
GO
```

现在创建登录触发器来限制新登录名从上午7:00到下午5:00的登录：

12

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
USE master
GO

CREATE TRIGGER trg_logon_attempt
ON ALL SERVER
WITH EXECUTE AS 'sa'
FOR LOGON
AS
BEGIN

IF ORIGINAL_LOGIN()='nightworker' AND
   DATEPART(hh,GETDATE()) BETWEEN 7 AND 17
   BEGIN
      ROLLBACK
      INSERT ExampleAuditDB.dbo.RestrictedLogonAttempt
      (LoginNM, AttemptDT)
      VALUES (ORIGINAL_LOGIN(), GETDATE())
   END
END
GO
```

390 现在尝试以nightworker登录名在指定时间范围内登录，你将看到如下的错误消息：

```
由于执行触发器，登录名'nightworker'的登录失败。
```

在尝试登录之后，我将查询审核表查看是否跟踪到了登录：

```
SELECT LoginNM, AttemptDT
FROM ExampleAuditDB.dbo.RestrictedLogonAttempt
```

这个查询返回如下结果（基于你自己执行这个操作的时间，结果会有所不同）：

```
LoginNM          AttemptDT
nightworker      2008-2-26 11:37:15.127
```

解析

使用登录触发器可以在SQL Server实例的身份验证之后生成实际会话之前限制和跟踪登录活动。如果想在SQL Server特性集之上应用自定义的业务规则到登录上，可以使用登录触发器来实现它。

在这个技巧中，创建了测试登录名、新的审核数据库以及跟踪登录尝试的审核表。在master数据库中，创建了登录触发器。一点点的来看代码，注意到使用ALL SERVER来设置触发器执行的范围：

```
CREATE TRIGGER trg_logon_attempt
ON ALL SERVER
```

使用EXECUTE AS子句来定义执行触发器的权限。我本来可以使用（权限）更小的登录名——例如，拥有审核表写权限的登录名：

```
WITH EXECUTE AS 'sa'
```

FOR LOGON关键字指定了创建的触发器类型：

```
FOR LOGON
AS
```

然后BEGIN关键字标志触发器逻辑主体的开始：

```
BEGIN
```

然后运算登录尝试的原始安全上下文。在这个例子中，我只关心登录名是nightworker时的强制逻

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

辑：

```
IF ORIGINAL_LOGIN()='nightworker' AND
```

在这个逻辑中包括对一天中小时的运算。如果当前时间在上午7点到下午5点之间，则将执行两个动作：

```
DATEPART(hh,GETDATE()) BETWEEN 7 AND 17
BEGIN
```

第一个动作是回滚登录尝试：

```
ROLLBACK
```

第二个动作是跟踪登录尝试的审核表：

```
INSERT ExampleAuditDB.dbo.RestrictedLogonAttempt
(LoginNM, AttemptDT)
VALUES (ORIGINAL_LOGIN(), GETDATE())
   END
END
GO
```

391

再次提醒你登录触发器代码的逻辑很重要。不当的登录可以引发意料之外的结果。而且如果登录触发器没有按你希望的进行工作，一定要检查最后的SQL登录。登录触发器尝试被回滚而且写入到SQL日志中。如果在触发器中漏写了一些代码，例如，如果没有适当地指定RestrictedLogonAttempt的完整表名，SQL日志会显示错误消息“对象名‘dbo.RestrictedLogonAttempt’无效”。

提示 不要忘记在完成测试之后移除这个触发器。在master数据库中执行DROP TRIGGER trg_logon_attempt ON ALL SERVER来删除它。

12.2.4 查看 DDL 触发器元数据

在这个技巧中，会演示获取DDL触发器元数据。

第一个示例查询sys.triggers目录视图，返回相关的数据库范围（database-scoped）的触发器名和触发器的启用或禁止状态：

```
USE AdventureWorks
GO
-- 显示当前数据库中的DDL触发器
SELECT name TriggerNM, is_disabled
FROM sys.triggers
WHERE parent_class_desc = 'DATABASE'
ORDER BY OBJECT_NAME(parent_id), name
```

12

这个查询返回如下的（部分）结果：

```
TriggerNM                    is_disabled
ddlDatabaseTriggerLog        1
```

下面的示例查询sys.server_triggers和sys.server_trigger_events系统目录视图来获取服务器范围（server-scoped）的DDL触发器列表。这个查询返回了DDL触发器名、触发器的类型（T-SQL或CLR）、触发器的禁止状态以及导致触发器触发的事件（对于每个触发器基于的事件你都会看到一行）：

```
SELECT name, s.type_desc SQL_or_CLR,
is_disabled, e.type_desc FiringEvents
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FROM sys.server_triggers s
INNER JOIN sys.server_trigger_events e ON
   s.object_id = e.object_id
```

这个查询基于前面创建的服务器级别的触发器返回数据：

```
name                SQL_or_CLR      is_disabled     FiringEvents
trg_logon_attempt   SQL_TRIGGER     0               LOGON
```

392 要显示数据库范围的DDL触发器的T-SQL定义，可以查询sys.sql_modules系统目录视图：

```
SELECT t.name, m.Definition
FROM sys.triggers AS t
INNER JOIN sys.sql_modules m ON
t.object_id = m.object_id
WHERE t.parent_class_desc = 'DATABASE'
```

要显示服务器范围的DDL触发器，可以查询sys.server_sql_modules和sys.server_triggers系统目录视图：

```
SELECT t.name, m.definition
FROM sys.server_sql_modules m
INNER JOIN sys.server_triggers t ON
   m.object_id = t.object_id
```

解析

技巧中的第一个查询使用sys.triggers系统目录视图返回数据库范围的触发器列表。为了只显示DDL数据库范围的触发器，就必须限制parent_class_desc的值为DATABASE。第二个查询用于返回服务器范围的触发器列表以及关联的触发器事件。对于这种情况，我们查询sys.server_triggers和sys.server_trigger_events系统目录视图。

第三个查询用于返回数据库范围的触发器的T-SQL定义，通过限制sys.triggers到sys.sql_modules系统目录视图来实现。要返回服务器范围的触发器的T-SQL定义，我们查询sys.server_sql_modules和sys.server_ triggers系统目录视图。

12.3　管理触发器

下面的一组技巧演示如何修改、删除、启用、禁止和控制触发器各个选项。将演示的一些命令包括：修改触发器定义的ALTER TRIGGER、从数据库中移除触发器的DROP TRIGGER、设置触发器递归选项的ALTER DATABASE、控制触发器嵌套的sp_configure以及设置触发器的触发次序的sp_settriggerorder。

12.3.1　修改触发器

可以使用ALTER TRIGGER命令来修改既有的DDL或DML触发器。ALTER TRIGGER和关联的DML或者DDL CREATE TRIGGER语法有相同的参数。

在本例中，我们会修改前面技巧中创建的触发器（在本章前面删除了这个触发器，因此你可以重新创建它来在这里演示）。这次不会限制用户新建登录名，而是允许登录事件，之后是一条警告和对审核表进行的INSERT操作：

```
USE master
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ALTER TRIGGER srv_trg_RestrictNewLogins
ON ALL SERVER
FOR CREATE_LOGIN
AS

SET NOCOUNT ON

PRINT 'Your login creation is being monitored.'

INSERT AdventureWorks.dbo.ChangeAttempt
(EventData, DBUser)
VALUES (EVENTDATA(), USER)

GO
```

393

解析

通过ALTER TRIGGER可以修改既有的DDL或DML触发器。ALTER TRIGGER的参数和CREATE TRIGGER的一样。然而，你不能使用它来修改实际的触发器名，所以在本例中，触发器名和DDL触发器真实的行为已经不相符了（在这里只是监视，而不再限制新的登录名）。

12.3.2 启用和禁止表触发器

有时候，如果触发器引发了问题需要调试，或者如果需要导入或恢复不能触发触发器的数据，则必须禁止它。在这个技巧中，会演示如何使用DISABLE TRIGGER命令来禁止触发器触发，以及如何使用ENABLE TRIGGER来重新启用触发器。

DISABLE TRIGGER的语法如下：

```
DISABLE TRIGGER [ schema . ] trigger_name
ON { object_name | DATABASE | SERVER }
```

启用触发器的语法如下：

```
ENABLE TRIGGER [ schema_name . ] trigger_name
ON { object_name | DATABASE | SERVER }
```

表12-5详细描述了这个命令的参数。

表12-5 ENABLE和DISABLE触发器参数

参 数	描 述
[schema_name .]trigger_name	可选的架构名和必需的、你希望禁止的用户定义的触发器名
object_name \| DATABASE \| SERVER	object_name是触发器绑定的表或视图（如果它是DML触发器的话）。如果触发器是DDL数据库范围的触发器则使用DATABASE，如果触发器是DDL服务器范围的触发器则使用SERVER

这个示例从创建一个触发器（默认是开启的）开始，对于HumanResources.Department表的INSERT操作，它会打印一个消息返回给连接：

```
USE AdventureWorks
GO

CREATE TRIGGER HumanResources.trg_Department
ON HumanResources.Department
AFTER INSERT
AS
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
PRINT 'The trg_Department trigger was fired'
GO
```

394

然后使用DISABLE TRIGGER命令来禁止触发器：

```
DISABLE TRIGGER HumanResources.trg_Department
ON HumanResources.Department
```

因为触发器被禁止了，所以下面的INSERT执行后不会返回打印消息：

```
INSERT HumanResources.Department
(Name, GroupName)
VALUES ('Construction', 'Building Services')
```

这个查询返回：

```
(1行受影响)
```

然后，使用ENABLE TRIGGER命令启用触发器：

```
ENABLE TRIGGER HumanResources.trg_Department
ON HumanResources.Department
```

现在再尝试INSERT的话，触发器会触发，返回一个消息给连接：

```
INSERT HumanResources.Department
(Name, GroupName)
VALUES ('Cleaning', 'Building Services')
```

这个查询返回：

```
The trg_Department trigger was fired

(1行受影响)
```

解析

这个技巧从创建一个新的触发器开始，当有新行插入到HumanResources.Department表的时候它就打印一条语句。

在创建了触发器后，使用DISABLE TRIGGER来禁止它触发（但是触发器的定义仍然保留在数据库中）：

```
DISABLE TRIGGER HumanResources.trg_Department
ON HumanResources.Department
```

然后执行了一次没有触发触发器的插入操作。接着执行ENABLE TRIGGER命令，并尝试另外一次插入，这次触发器触发了。

12.3.3　限制触发器嵌套

当触发器触发后执行的行动（例如，对另外一个表进行插入）会触发另外一个触发器，那个触发器然后又会触发另外一个触发器的时候就发生了触发器嵌套。SQL Server通过嵌套的最大层次来防止产生触发器的无限循环触发，这个层次是32层。

也可以修改SQL Server实例来禁止触发器嵌套。禁止nested triggers选项能防止任何AFTER触发器再引起其他触发器的触发。

395

本例演示了如何禁止和启用这个行为：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
USE master
GO

-- 禁止嵌套
EXEC sp_configure 'nested triggers', 0
RECONFIGURE WITH OVERRIDE
GO

-- 启用嵌套
EXEC sp_configure 'nested triggers', 1
RECONFIGURE WITH OVERRIDE
GO
```

这个查询返回：

```
配置选项'nested triggers'已从1更改为0。
请运行RECONFIGURE语句进行安装。
配置选项'nested triggers'已从0更改为1。
请运行RECONFIGURE语句进行安装。
```

解析

这个技巧使用sp_configure系统存储过程在服务器级别修改嵌套的触发器行为。要完全禁止嵌套，可以执行sp_configure的nested trigger服务器选项，后面跟参数0表示禁止嵌套：

```
EXEC sp_configure 'nested triggers', 0
RECONFIGURE WITH OVERRIDE
GO
```

由于服务器选项包含当前的配置和运行时配置值，RECONFIGURE WITH OVERRIDE命令用于更新运行时值使之立即生效。

要再次启用嵌套，可以在技巧的第二个批处理中把这个服务器选项设置回1。

注解 更多有关配置服务器选项的信息请参阅第21章。

12.3.4 控制触发器递归

如果触发器触发后执行的行为会导致相同的表触发器再次触发，那么这种触发器嵌套就被认为是递归的。当触发器的触发影响其他表的时候，如果触发器还会影响原始表，引起原来的触发器再次触发，这样也会发生递归。

可以通过配置RECURSIVE_TRIGGERS数据库选项来控制是否允许递归。如果允许递归，则AFTER触发器仍然会受到32层嵌套限制的影响，以防止无限循环发生。

本例演示了启用和禁止这个选项：

```
-- 允许递归
ALTER DATABASE AdventureWorks
SET RECURSIVE_TRIGGERS ON

-- 查看数据库设置
SELECT is_recursive_triggers_on
FROM sys.databases
WHERE name = 'AdventureWorks'
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
-- 防止递归
ALTER DATABASE AdventureWorks
SET RECURSIVE_TRIGGERS OFF

-- 查看数据库设置
SELECT is_recursive_triggers_on
FROM sys.databases
WHERE name = 'AdventureWorks'
```

这个查询返回：

```
is_recursive_triggers_on
1

is_recursive_triggers_on
0
```

解析

ALTER DATABASE用于配置数据库级别的选项，包括是否允许在数据库内递归触发触发器。通过设置RECURSIVE_TRIGGERS ON来启用这个选项：

```
ALTER DATABASE AdventureWorks
SET RECURSIVE_TRIGGERS ON
```

然后使用sys.databases系统目录视图来查询这个选项，它显示了当前数据库选项的is_recursive_triggers_on字段（1表示开启，0表示关闭）：

```
SELECT is_recursive_triggers_on
FROM sys.databases
WHERE name = 'AdventureWorks'
```

然后通过设置选项为OFF来禁止触发器递归，之后再次通过查询sys.databases来确认。

注解　更多有关ALTER DATABASE和数据库选项的信息请参阅第22章。

12.3.5　设置触发器触发次序

通常应该把对同样事件作出响应的触发器封装到一个触发器中。这样能增加触发器的可管理性和
397 可支持性，因为更容易找到需要的代码，然后能进行相应的调试。你同样能避免指定哪个触发器先运行的问题。然而，确实可以在一个表上定义多个触发器，引用相同的DML类型（例如多个INSERT触发器）。在同一个数据库或服务器范围的事件或事件组上同样也能设置多个DDL触发器。

如果确信必须有独立的触发器引用相同的数据库对象（可能你已经增加了一个触发器以不和第三方的代码重叠），并且触发器触发的次序对你来说很重要，那么应该使用sp_settriggerorder系统存储过程来进行配置。

sp_settriggerorder的语法如下：

```
sp_settriggerorder [ @triggername = ] '[ triggerschema. ] triggername'
      , [ @order = ] 'value'
      , [ @stmttype = ] 'statement_type'
      [ , [ @namespace = ] { 'DATABASE' | 'SERVER' | NULL } ]
```

表12-6详细描述了这个命令的参数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表12-6 sp_settriggerorder参数

参 数	描 述
'[triggerschema.] triggername'	这个参数定义了可选的架构拥有者和需要排序的用户定义的触发器名
[@order =] 'value'	可以是First、None或Last。任何介于第一个触发和最后一个触发之间的触发器会以随机次序触发
[@stmttype =] 'statement_type'	这个参数指定需要排序的触发器类型。例如，INSERT、UPDATE、DELETE、CREATE_INDEX、ALTER_INDEX，等等
[@namespace =] { 'DATABASE' \| 'SERVER' \| NULL}	这个参数指定是否为DDL触发器，如果是的话，是数据库范围的还是服务器范围的

这个技巧会创建一个测试表并为之增加3个DML INSERT触发器。然后使用sp_settriggerorder来定义触发次序：

```
CREATE TABLE dbo.TestTriggerOrder
   (TestID int NOT NULL)
GO

CREATE TRIGGER dbo.trg_i_TestTriggerOrder
ON dbo.TestTriggerOrder
AFTER INSERT
AS
PRINT 'I will be fired first.'
GO

CREATE TRIGGER dbo.trg_i_TestTriggerOrder2
ON dbo.TestTriggerOrder
AFTER INSERT
AS
PRINT 'I will be fired last.'
GO

CREATE TRIGGER dbo.trg_i_TestTriggerOrder3
ON dbo.TestTriggerOrder
AFTER INSERT
AS
PRINT 'I won''t be first or last.'
GO

EXEC sp_settriggerorder 'trg_i_TestTriggerOrder', 'First', 'INSERT'
EXEC sp_settriggerorder 'trg_i_TestTriggerOrder2', 'Last', 'INSERT'

INSERT dbo.TestTriggerOrder
(TestID)
VALUES (1)
```

这个查询返回：

```
I will be fired first.
I won't be first or last.
I will be fired last.
```

解析

这个技巧从创建一个单列的测试表开始。然后为它加入3个DML INSERT触发器。使用sp_settriggerorder来定义最先和最后触发的触发器：

398

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
EXEC sp_settriggerorder 'trg_i_TestTriggerOrder', 'First', 'INSERT'
EXEC sp_settriggerorder 'trg_i_TestTriggerOrder2', 'Last', 'INSERT'
```

然后对表进行INSERT，触发器消息按照期望的次序返回。

要重申一点，如果可以的话，请在表上只使用一个触发器。如果必须创建具有相同类型的多个触发器，而且触发器在遇到错误的时候也包含ROLLBACK功能，请确保让最有可能失败的触发器先执行，这样的话只有第一个触发的触发器需要执行，以防止触发不必要的触发器以及回滚事务。

12.3.6　删除触发器

删除触发器的语法根据触发器类型（DML或DDL）的不同而不同。

删除DML触发器的语法如下：

```
DROP TRIGGER schema_name.trigger_name [ ,...n ]
```

表12-7详细描述了这个命令的参数。

表12-7　DROP TRIGGER参数（DML）

参　数	描　述
schema_name . trigger_name	触发器的拥有者架构名和要从数据库中移除的DML触发器名

399

删除DDL触发器的语法如下：

```
DROP TRIGGER trigger_name [ ,...n ]
ON { DATABASE | ALL SERVER }
```

表12-8详细描述了这个命令的参数。

表12-8　DROP TRIGGER参数（DDL）

参　数	描　述
trigger_name	定义要从数据库（对于数据库范围的DDL触发器）或SQL Server实例（对于服务器范围的触发器）中移除的DDL触发器名
DATABASE \| ALL SERVER	定义要移除的是数据库范围的DDL触发器（DATABASE），还是服务器范围的触发器（ALL SERVER）

对于DDL和DML语法语句，[,...n]语法段都表示可以同时移除多个触发器。

本例演示了删除DML和DDL触发器：

```
--删除一个DML触发器
DROP TRIGGER dbo.trg_i_TestTriggerOrder

--删除多个DML触发器
DROP TRIGGER dbo.trg_i_TestTriggerOrder2, dbo.trg_i_TestTriggerOrder3

--删除DDL触发器
DROP TRIGGER db_trg_RestrictINDEXChanges
ON DATABASE
```

解析

在这个技巧中，使用DROP TRIGGER命令来显式删除DML和DDL触发器。在删除表或视图的时候也会删除绑定到它们的所有DML触发器。如果每个触发器都是使用相同的ON子句来创建的，也可以用同一个DROP命令一次删除多个触发器。

400

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 13 章 CLR 集成 13

尽管本书重点在T-SQL语言上，但公共语言运行时（CLR）和T-SQL之间很多地方有交错。本章会介绍这些内容并且给出一些技巧。

在一些人眼里，在数据库里包含CLR是SQL Server最大的进步。有了CLR，开发者就不用必须使用T-SQL来写诸如存储过程、函数和触发器等过程式数据库对象。现在，它们可以使用任意一种.NET语言（VB.NET、C#、C++等）来创建这些对象，并把它们编译成.NET程序集（assembly）。这些程序集部署在数据库内，由CLR来执行，也就会保存在SQL Server内存空间中。

SQL Server数据库的传统编程语言T-SQL，对于大量的数据操作来说是强大的语言，但是它的计算复杂度有限制。对于那些在数据库中的复杂操作，开发者只能求助于以复杂闻名的C++写的扩展过程，或者创建非常长的、难以使用的存储过程代码。

理论上说，CLR集成提供了“两全其美”的方案。我们的代码能承载于数据库的安全环境、代理内存管理、垃圾收集和对强大数据库引擎的线程支持，并且能使用.NET强大的计算能力、高级数据类型支持和丰富的内建类库。

尽管本书重点是T-SQL功能，我仍然会介绍创建程序集，把它们导入数据库以及关联数据库对象的一些基本方法。我首先会一步步介绍基本步骤，然后会深入介绍各种不同的CLR数据库对象类型的变化。本章的讨论和技巧包括如下内容：

- 讨论什么时候应该在SQL Server中使用程序集，什么时候不应该；
- 可用的SQL Server CLR数据库对象以及如何创建它们；
- 通过一个个技巧带你来创建CLR存储过程；
- 创建CLR标量用户定义函数；
- 创建CLR触发器；
- 从数据库中查看、修改和移除程序集。

然而，在本章的一开始，会先总体介绍一下公共语言运行时（CLR）。

13.1 CLR 概述

在进一步讨论SQL Server集成之前，我想先为那些刚接触.NET框架（.NET framework）的读者介绍一些基础知识。首先，.NET框架是用于构建微软Windows程序和服务的可编程平台。这个框架能用于创建Windows表单、Web服务和ASP.NET应用程序，等等。框架的主要部分包括CLR、框架类和库（包含一些可为应用程序使用的可重用的功能和编程模型）以及ASP.NET（用于构建基于Web的应用程序）。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注解　.NET编程需要微软.NET框架。这就是为什么微软Windows .NET框架是安装SQL Server的必要条件。

公共语言运行时（CLR）是.NET程序实际运行和托管的环境。CLR用于执行.NET程序、管理内存以及维护程序元数据。在前言中提到过，从SQL Server 2005到SQL Server 2008，CLR承载于SQL Server进程中。这意味着，SQL Server进程中的保留空间负责处理内存管理、安全以及执行的上下文。

写托管.NET代码的时候（在CLR中执行和托管的代码），程序集是用来实现功能的打包后的DLL或可执行文件。你可以把程序集关联到诸如触发器、存储过程、用户定义函数（UDF）等各种数据库对象。使用基于CLR的数据库提供了广阔的功能，可以实现复杂的计算、访问外部数据源的数据、与其他应用程序整合以及解决T-SQL不能处理的问题。

你可以用最习惯的.NET语言来写程序集——Visual Basic.NET和C#是最常用的两种。你可以选择喜欢的.NET语言，原因之一是代码首先编译成中间语言（intermediate language，IL）形式。CLR读取和执行的是IL形式。使用C#或VB.NET（Visual Basic.NET的简写）写的代码实现相同的功能，它们最终都会变为中间语言指令，两者的中间语言看上去都差不多。

除了编程语言之外，同样可以选择如何开发代码。一个很好的选择就是Visual Studio，它包含一些框架，能使SQL Server数据库CLR对象的创建变得简单。当然，你也可以不必使用Visual Studio，还有很多开源的.NET开发工具能从网上下载到。你也可以在Windows记事本中硬编码.NET应用程序。尽管这种方法对开发项目不是很理想，但它不需要额外的软件，本章也会使用这个方法。我使用这个原始的方法是为了重点讨论CLR和T-SQL的集成，而不是深入考虑Visual Studio的一些特性。

13.2　什么时候使用（不使用）程序集

CLR以及.NET框架和SQL Server整合的公告在经验丰富的SQL Server的专业人员和开发人员中引起了矛盾的心情。一方面，人们对.NET数据库完全替代T-SQL满怀憧憬。另一方面，那些没有.NET编程背景的数据库管理员和开发者则很担心，早早地就做出不使用这个新特性的决定。

402

首先也是最明确的一点就是，基于.NET的数据库对象不可能取代T-SQL创建的数据库对象。T-SQL仍然有生命力。大多数的功能必须使用T-SQL来实现，而一些.NET构建模型和编程模型最终还是要转化为使用T-SQL。

有两个主要的理由来考虑使用CLR数据库对象。

- 有一些“数据相关的”业务逻辑以前不能在数据库内实现，必须使用现有的功能和T-SQL。因此，在客户端层或中间层创建了扩展存储过程或模块。
- 强制T-SQL来实现非常复杂或计算量极大的任务，产生了复杂的和不优雅的或低效率的T-SQL代码。

从某些方面说，使用.NET CLR来替代扩展存储过程是使用程序集的最佳应用。在之前版本的SQL Server中，如果需要为SQL Server添加之前没有的功能或者需要访问外部资源，一个通常的方案就是使用*扩展存储过程*。数据库用户就像使用普通存储过程一样调用扩展存储过程并且可以传参数给它们。扩展存储过程可以使用编程语言（比如C++）来写，然后转化成DLL文件。sp_addextendedproc系统存储过程用于创建新的存储过程并且把它绑定到DLL文件（必须在SQL Server实例中存在）。DLL文件不是导入到SQL Server数据库的，所以它必须存在于SQL Server实例的机器上。因为DLL在SQL Server中载入并使用，而且没有任何特殊的管理或保护，这样就增加了内存泄漏或性能问题的风险，根据DLL代码的编写方式，风险各不相同。如果DLL工作不正常，SQL Server也可能崩溃。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

CLR集成就是用于解决扩展存储过程的这些问题的。如果使用托管代码，不可能发生内存泄漏，安全性也与SQL Server环境完全整合。简而言之，使用程序集比使用扩展存储过程更安全。所以：

- 如果数据库应用程序必须执行不能（或者非常困难）使用T-SQL来实现的、很复杂的计算；
- 如果数据库应用程序需要访问在.NET框架中存在而在T-SQL中不存在的功能；
- 如果数据库应用程序需要访问外部数据源（Web服务、文件、系统设置），而T-SQL不能访问；

那么就可以考虑把程序集作为潜在的解决方案。如果在系统中的扩展存储过程中已经实现了此类功能，那么它们可能就是程序集整合的首选候选。

如果系统的其他层（不管是客户端还是中间层）有复杂的业务逻辑，那么你需要一个一个检查这些应用，确认把它们转移到数据库的程序集中是否是明智的选择。集成其他应用程序的数据库应用程序以及针对同一数据库的即席报表对于现在的应用程序来说是很普通的组件。如果业务规则和数据本身相关，那么把这个逻辑封装到数据库中很有意义，这样不同的数据消费者就不需要复制这些规则。

一件需要肯定的事是，CLR数据库对象不应该用来替代T-SQL既有的功能。对于大量数据的获取和修改操作来说，使用SELECT/INSERT/UPDATE/DELETE的基于集合的处理总是首选方案和最佳的执行方案。如果某个操作能使用T-SQL在数据库内有效执行，那么还是应该使用T-SQL而不是CLR方法。 403

13.3 CLR 对象概述

要在SQL Server中使用CLR支持，必须把程序集创建和编译成DLL，然后再把新的程序集导入（使用CREATE ASSEMBLY）SQL Server数据库。一旦集成到了数据库，可以随表、数据和其他数据库对象一起备份——因为它也像其他数据库对象一样。一旦加入了程序集，你就可以关联它到其他数据库对象，包括用户定义函数、存储过程、触发器、用户定义类型和聚合函数。

- *用户定义函数*。它们创建能访问.NET框架的计算类和外部资源的标量或表值函数。在本章的后面会看到在标量函数中使用普通表达式的示例（不能使用T-SQL的情况）。
- *存储过程*。这可能是最有创造性的SQL Server数据库对象。可以使用基于CLR的存储过程来替代扩展存储过程，利用.NET框架类来实现T-SQL不可能做到的大量的计算和外部资源的行为。
- *触发器*。使用它可以创建.NET组件来对数据操纵语言（INSERT/UPDATE/DELETE）或数据定义语言（CREATE、ALTER、DROP）作出响应。
- *用户定义类型*。使用它可以创建新的复杂数据类型（和基于既有数据类型的T-SQL用户定义类型不同）。CLR用户定义类型包括和.NET对象或类统一的方法和属性。这可能是SQL Server较有争议的扩充之一。因为一个类型的多个属性可能违背关系型数据库的基本设计原则。使用CLR用户定义类型可以实现数据验证和字符串格式化，这些对于T-SQL用户定义类型是不可能的。
- *用户定义聚合函数*。你不能使用T-SQL创建聚合函数。但可以创建基于CLR的用户定义聚合函数在SQL Server中创建新的聚合函数。用户定义聚合函数可以用来创建SQL Server不提供的复杂统计分析聚合，或者把多个字符串值收集到一个业务自定义结果中。

本章余下的部分重点讲解创建CLR存储过程、用户定义函数和触发器。因为它们和T-SQL的副本（在访问和执行方面）非常相似，因此也是本书最相关的部分。

13

13.4 创建 CLR 数据库对象

本节的一些技巧会带你一步步创建3个基于CLR的对象，即CLR存储过程、CLR UDF和CLR触发器。对于CLR存储过程，我会介绍4个子技巧来描述下面每个步骤。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 使用系统存储过程sp_configure来为SQL Server实例开启CLR功能。如果计划为CLR数据库对象使用EXTERNAL_ACCESS或UNSAFE权限，把你要使用CLR数据库对象的那个数据库设置成TRUSTWORTHY。
- 使用你喜欢的.NET语言和工具来创建程序集代码。例如，可以使用C#或VB.NET来创建程序集。使用Visual Studio创建CLR程序集会更简单，不过也可以使用简单的记事本和vsc.exe编译器。
- 把代码编译成DLL文件。
- 使用CREATE ASSEMBLY T-SQL命令来把新的程序集加载到数据库中。根据程序集的功能选择安全级别。尝试在SAFE或EXTERNAL_ACCESS安全级别下构建代码。这些级别为SQL Server实例提供了更好的稳定性，并且有助于避免不安全代码引发的潜在问题。

404

在这之后，会演示如何创建CLR标量用户定义函数，也会有相似的步骤（在一个技巧中），但是使用的是全新的程序集，代码也会有一些难懂的地方。最后，会介绍CLR触发器。

13.4.1　启用 SQL Server 的 CLR 支持

SQL Server安装好之后，CLR功能默认是禁止的。要允许使用CLR数据库对象，就必须使用sp_configure系统存储过程来配置'clr enabled'选项（有关这个系统存储过程的全部内容请参阅第21章）：

```
EXEC sp_configure 'clr enabled', 1
RECONFIGURE WITH OVERRIDE
GO
```

这个查询返回：

```
配置选项'clr enabled'已从0更改为1。
请运行RECONFIGURE语句进行安装。
```

如果计划使用需要EXTERNAL_ACCESS或UNSAFE安全权限的CLR数据库对象，必须把TRUSTWORTHY数据库选项设置为ON。例如：

```
IF NOT EXISTS (SELECT 1 FROM sys.databases WHERE name = 'BookStore')
BEGIN
   CREATE DATABASE BookStore
END
GO

ALTER DATABASE BookStore
SET TRUSTWORTHY ON
```

解析

这个示例演示了为SQL Server实例开启CLR功能。在执行命令后，CLR功能就立即开启了，无需重新启动SQL Server实例。然后我们为BookStore数据库启用TRUSTWORTHY选项，以便之后使用需要EXTERNAL_ACCESS和UNSAFE安全权限的CLR数据库对象（尽管这样，我们只会演示一个需要外部访问的CLR数据库对象，而不是演示一些不安全的东西）。

在下个技巧中，会演示使用VB.NET创建程序集。

13.4.2　为 CLR 存储过程写程序集

在这个技巧中，会演示为程序集创建代码。特别地，使用VB.NET代码来从外部文本文件读取数

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

据并且在结果集中输出文本文件。

在接触真实代码之前，首先需要讨论有关程序集本身的一些新概念。

目前为止我们讨论的CLR程序集似乎是用于一个一对一的数据库对象。其实，程序集包含的代码能用于一个或者多个CLR数据库对象。例如，在这个技巧中会使用的代码用于一个存储过程。然而，可以把几个子程序或者类型放到一个程序集中，为不同的CLR数据库对象所用。作为最佳实践，尝试把相关的功能组合到一个程序集中。如果各种函数或方法相互依赖的话，这非常重要。想象一下这种情况，有一组会交叉引用外部地图应用程序的功能。例如，程序集可以包含CLR存储过程用来返回驾驶方向的代码、CLR用户定义函数用来根据地址输入信息返回地图坐标的代码和新的用户定义CLR类型来包含各种详细地址。

另外一个需要理解的重要概念是程序集安全。使用托管代码时，必须考虑代码需要对某个资源有多少访问权限。在13.4.4节的技巧中你会看到，当把程序集载入SQL Server时，需要指示程序集需要的权限级别。你会有3个选择：SAFE、EXTERNAL_ACCESS和UNSAFE。本章稍后会详细描述。

这个程序集示例演示了使用VB.NET创建类和函数，它接受文件和路径名作为输入值，打开文件进行读取，最后把结果返回给调用它的SQL Server连接上下文。我会在“解析”部分中讨论这个脚本的元素：

```
Imports System.Data
Imports System.Data.Sql
Imports System.Data.SqlTypes
Imports Microsoft.SqlServer.Server
Imports System.IO

Public Class ReadFiles

Public Shared Sub Main(ByVal sFile As SqlString)

   Dim sReader As StreamReader = New StreamReader(sFile)
   Dim sLine As String
   Dim sPipe As SqlPipe = SqlContext.Pipe

   Do
   sLine = sReader.ReadLine()
 If Not sLine Is Nothing Then
  sPipe.Send(sLine)
End If
   Loop Until sLine Is Nothing

   sReader.Close()

   End Sub
End Class
```

解析

这个技巧的示例包含了一个类和函数，它们会关联到某个CLR存储过程。CLR数据库对象需要在程序集中存在某些命名空间，这样SQL Server才能引用程序集代码中内建的CLR程序集。例如，代码包含如下的命名空间：


```
Imports System.Data
Imports System.Data.Sql
Imports System.Data.SqlTypes
Imports Microsoft.SqlServer.Server
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

你也可以根据程序集需要的功能包含其他命名空间。例如，System.IO命名空间包含需要从文件统文件中读取和写入的函数：

```
Imports System.IO
```

这个示例又声明了一个叫ReadFiles的公共类：

```
Public Class ReadFiles
```

接着，是包含一个参数字符串值（在这里是要读取的文件名和路径）的公共共享函数：

```
Public Shared Sub Main(ByVal sFile As SqlString)
```

注意到sFile输入参数被定义成SqlString类型。使用CLR程序集时，需要理解关联到某个SQL CLR .NET数据类型的SQL Server数据类型。表13-1列出了某些可用的CLR数据类型和对应的SQL Server数据类型（注意，对于重复的类型你可以选择使用）。

表13-1　把SQL Server数据类型转化为CLR数据类型

CLR数据类型	SQL Server数据类型
SqlBytes	varbinary,binary
SqlBinary	varbinary,binary
SqlChars（对于数据访问和获取比较适用）	nvarchar,nchar
SqlString（对于字符串操作比较适用）	nvarchar,nchar
SqlGuid	uniqueidentifier
SqlBoolean	bit
SqlByte	tinyint
SqlInt16	smallint
SqlInt32	int
SqlInt64	bigint
SqlMoney	smallmoney, money
SqlDecimal	decimal, numeric
SqlSingle	real
SqlDouble	float
SqlDateTime	smalldatetime, datetime, date, datetime2
TimeSpan	time
SqlXml	xml

继续看这个示例，注意到声明的StreamReader对象传入了文件名和路径。StreamReader类用于从文件读取文本数据。因为它不是T-SQL函数，所以按照一般情况不能在代码中引用这个函数。CLR程序集可以使用SQL Server数据库的这些.NET命令：

407

```
Dim sReader As StreamReader = New StreamReader(sFile)
```

创建一个字符串变量来保存文件中的一行数据：

```
Dim sLine As String
```

接着，我使用了两个类：SqlPipe和SqlContext：

```
Dim sPipe As SqlPipe = SqlContext.Pipe
```

SqlPipe对象用来把0行或更多的行返回给已连接的调用连接。所以，如果执行CLR存储过程应该能得到一个列表结果（和SELECT查询相似）。这里使用了SqlPipe对象的Send方法。SqlContext类维持和访问SQL Server调用方的上下文，也就是说如果执行存储过程，SqlContext知道这是我的行为，那么

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

结果属于我的客户端。SqlPipe使用SqlContext的Pipe方法[1]根据用户连接的SqlContext得到。

接着创建了Do循环（和T-SQL的WHILE相似）来遍历文件的每一行，直到没有任何行：

```
Do
```

使用StreamReader对象的ReadLine方法来把文件的第一行赋值给sLine变量：

```
sLine = sReader.ReadLine()
```

如果文件中的行有值，那行的值就会通过SqlPipe对象的Send方法被送回SQL Server连接：

```
 If Not sLine Is Nothing Then
  sPipe.Send(sLine)
End If
```

然而，一旦文件结束了，Do循环也就结束了，然后关闭文件的连接：

```
Loop Until sLine Is Nothing

sReader.Close()
```

关闭方法和类定义来结束程序集。

```
   End Sub
End Class
```

现在，已经使用VB.NET编写了程序集，那么们可以进入下一步了，就是把程序集代码编译成DLL文件，这样才能导入SQL Server。先做些准备工作，创建一个叫C:\Apress\Recipes\CLR\的文件目录，然后把文件保存为ReadFiles.vb。

13.4.3 把程序集编译成 DLL 文件

我们使用vbc.exe来编译程序集文件，而没有使用Visual Studio。vbc.exe编译器能在SQL Server实例所在机器的最新版本的C:\WINDOWS\MICROSOFT.NET\framework\目录中找到。

在这个示例中，使用下面的命令来创建DLL程序集，通过在命令行提示下执行vbc可执行文件来把ReadFiles.vb转化成DDL程序集（要注意的是因为你的SQL实例目录路径不同，sqlaccess.dll的引用也会有所不同）：

408

```
vbc /t:library /out:C:\Apress\Recipes\CLR\ReadFiles.DLL /r:"C:\Program
 Files\Microsoft SQL Server\MSSQL10.AUGUSTUS\MSSQL\Binn\sqlaccess.dll"
"C:\Apress\Recipes\CLR\ReadFiles.vb"
```

解析

在本技巧中执行vbc.exe可执行文件在C:\Apress\Recipes\CLR目录下创建DLL文件。这样就可以在SQL Server中创建程序集，接下来我们会讨论如何实现。

13.4.4 把程序集载入 SQL Server

使用CREATE ASSEMBLY命令可以把新的程序集载入SQL Server 2008数据库。

本例中使用的基本语法如下所示：

```
CREATE ASSEMBLY assembly_name
[ AUTHORIZATION owner_name ]
FROM { '[\\computer_name\]share_name\[path\]manifest_file_name'
  | '[local_path\]manifest_file_name'|
```

① 这里的Pipe是属性。——译者注

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
{ varbinary_literal | varbinary_expression }]
[ WITH PERMISSION_SET = { SAFE | EXTERNAL_ACCESS | UNSAFE } ]
```

表13-2描述了这个命令的参数。

表13-2　CREATE ASSEMBLY参数

参　数	描　述
assembly_name	这个参数定义了新的数据库程序集名
owner_name	这个参数定义了程序集的用户或者角色拥有者
'[\\computer_name\]share_name\[path\]manifest_file_name'\|'[local_path\]manifest_file_name' \|	这个参数定义了要加载的程序集的路径和文件名
varbinary_literal \| varbinary_expression	组成程序集的二进制数据可以代替实际文件来传入命令
SAFE \| EXTERNAL_ACCESS \| UNSAFE	这是程序集使用的安全权限级别，这部分内容一开始讨论过

程序集的安全权限级别需要认真考虑一下。SAFE权限能运行只需要访问本地SQL Server实例的代码。使用这个默认模式，程序集不能访问网络、外部文件（甚至SQL Server实例所在机器的文件）、注册表或环境变量。EXTERNAL_ACCESS权限允许访问网络、外部文件、注册表、环境变量和Web服务。SAFE和EXTERNAL_ACCESS模式都有一定的内部安全级别。这些内部措施包括保护其他应用程序的内存空间，
409 以及限制任何可能损害SQL Server实例的行为。

UNSAFE权限和本章开始讨论的扩展存储过程最相似。这个级别的权限不会对程序集如何访问资源作任何限制，允许破坏内存空间或执行会影响SQL Server实例稳定性的行为。你可能也知道，应该尽量避免使用这个权限级别。只有在必要时，且能确保程序集彻底测试过而且没有负面影响的情况下才去使用它。

继续本部分创建CLR存储过程的示例，根据ReadFiles.DLL来创建新的程序集，因为程序集需要从文件系统读取数据，所以使用EXTERNAL_ACCESS选项：

```
USE BookStore
GO

CREATE ASSEMBLY ReadFiles FROM 'C:\Apress\Recipes\CLR\ReadFiles.DLL'
WITH PERMISSION_SET = EXTERNAL_ACCESS
GO
```

解析

创建新的程序集后，需要把真实的程序集内容加载到数据库中。这意味着数据库备份同样会备份包含在其中的程序集。在示例中，根据程序集DLL文件来创建叫ReadFiles的新程序集。权限设置成EXTERNAL_ACCESS，因为程序集需要从文件读取数据并作为结果集将它返回给SQL Server调用方。

把程序集导入SQL Server还不足以使用这个功能。必须再把程序集关联到CLR数据库对象。下面的技巧会演示如何实现。

13.4.5　创建 CLR 存储过程

CLR数据库对象的创建和一般的T-SQL版本差不多，只是过程的定义需要引用程序集。下面每个命令都有EXTERNAL NAME的CLR选项：

- CREATE PROCEDURE;
- CREATE FUNCTION;

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- CREATE TRIGGER;
- CREATE TYPE。

除此之外，用于创建用户定义SQL Server聚合函数的CREATE AGGREGATE命令不能使用T-SQL来写，只能用于和.NET程序集联合。

提示 在SQL Server 2008中，CLR用户定义类型的大小不再被限制为8 000字节。

创建基于CLR的存储过程、用户定义函数或触发器的具体扩展语法如下：

```
EXTERNAL NAME assembly_name.class_name.method_name
```

要创建一个新的CLR数据类型或聚合函数，只需要引用程序集名和类名：

410

```
EXTERNAL NAME assembly_name [ .class_name ]
```

这个示例演示使用CREATE PROCEDURE命令的EXTERNAL NAME扩展来新建CLR存储过程，并把它映射到前面技巧新建的程序集：

```
CREATE PROCEDURE dbo.usp_FileReader
(@FileName nvarchar(1024))
AS EXTERNAL NAME ReadFiles.ReadFiles.Main
GO
```

ReadFiles出现两次，分别是CLR程序集名和VB.NET代码段的类名。

一旦创建后，就可以像一般T-SQL定义的存储过程那样执行CLR存储过程。继续这个示例，在存储过程的结果中返回SQL Server错误日志文件的内容(查看不被当前SQL Server实例使用的错误日志)：

```
EXEC dbo.usp_FileReader
N'C:\Program Files\Microsoft SQL Server\MSSQL10.AUGUSTUS\MSSQL\LOG\ERRORLOG.1'
```

这个查询返回ERRORLOG文件的内容作为结果集（部分内容)：

```
...
2007-10-14 08:09:18.91 Server      Using locked pages for buffer pool.
2007-10-14 08:09:18.92 Server      Using dynamic lock allocation.
Initial allocation of 2500 Lock blocks and 5000 Lock Owner blocks per node.
This is an informational message only. No user action is required.
2007-10-14 08:09:19.16 Server      Node configuration: node 0: CPU mask:
0x0000000000000003 Active CPU mask: 0x0000000000000003.
This message provides a description of the NUMA configuration for this computer.
This is an informational message only. No user action is required.
2007-10-14 08:09:19.27 Server      FILESTREAM: effective level = 0, configured level
= 0, file system access share name = 'CAESAR'.
2007-10-14 08:09:19.27 Server      Attempting to initialize
Microsoft Distributed Transaction Coordinator (MS DTC).
This is an informational message only. No user action is required.
2007-10-14 08:09:20.65 Server      Attempting to recover in-doubt distributed
transactions involving Microsoft Distributed Transaction Coordinator (MS DTC).
This is an informational message only. No user action is required.
2007-10-14 08:09:20.65 Server      Database mirroring has been enabled on this
instance of SQL Server.
2007-10-14 08:09:20.67 spid7s      Starting up database 'master'
...
```

一旦创建后，也可以通过使用ALTER或DROP命令来修改或删除数据库CLR对象的数据库对象类型。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

这个技巧演示了如何创建CLR存储过程。存储过程需要的参数取决于.NET程序集方法需要的参数，在这里，ReadFiles程序集的Main方法需要一个字符串参数作为要读取的文件和路径名，所以在存储过程引用中使用了一个@FileName nvarchar数据类型。在EXTERNAL NAME子句中引用了ReadFiles程序集，其后面是ReadFiles类和Main方法。

411

使用.NET框架，存储过程就能访问外部资源来遍历文件内容。有了CLR集成，SQL Server就能实现.NET框架提供的功能。

13.4.6 创建 CLR 标量用户定义函数

在本章开始的介绍中已经提到过，使用它执行非常复杂的计算操作时才会更好地得益于CLR。为不关注从SQL Server中获取数据的标量UDF函数使用CLR通常会比T-SQL版本执行得更好。CLR标量UDF对于不能使用T-SQL实现的操作同样有用（例如，访问外部数据或使用T-SQL不提供的.NET库功能）。

在这个示例中，我们创建一个程序集，它包含一个类和方法用于和CLR用户定义标量函数一起使用。我会用到.NET框架的System.Text.RegularExpressions命名空间。它包含了一个叫Regex的类，能让我们根据某个分隔符把某个字符串分割成数组值。正则表达式功能通常用于模式匹配，SQL Server 2008没有提供这个功能，但是有了CLR集成，你就可以安全高效地使用VB.NET的正则表达式库。

这个示例创建了一个接受3个参数的标量UDF。第一个参数指定要分隔的字符串值，第二个参数指定用于分隔字符串的分隔符。第三个参数是要从数组中选择的值。我会在“解析”部分详细介绍代码，但是现在你可以使用vbc.exe编译这个示例的代码：

```
Imports System.Data
Imports System.Data.Sql
Imports System.Data.SqlTypes
Imports Microsoft.SqlServer.Server
Imports System.Text.RegularExpressions

Public Class SQLArrayBuilder

Public Shared Function ChooseValueFromArray(ArrayString as String,
ArrayDelimiter as String, ArrayItemSelection as SqlInt16) as SqlString

Dim NewArrayString as String() = Regex.Split(ArrayString, ArrayDelimiter)

Dim NewArrayItemSelection as SqlInt16=ArrayItemSelection-1

Dim ReturnString as SQLString = NewArrayString(NewArrayItemSelection)


Return ReturnString

End Function

End Class
```

在保存VB文件SQLArrayBuilder.vb并使用vbc编译这个程序集后，就可以把它导入到数据库。因为程序集没有访问外部资源，所以可以使用SAFE权限级别：

```
CREATE ASSEMBLY SQLArrayBuilder FROM 'C:\Apress\Recipes\CLR\SQLArrayBuilder.DLL'
WITH PERMISSION_SET = SAFE
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

412

```
GO
```

接着，把新的程序集关联到一个标量用户定义函数。注意到，除了在AS后写EXTERNAL NAME关键字来指定程序集、类和函数，其他和T-SQL命令一样：

```
CREATE FUNCTION dbo.CountSalesOrderHeader
(@ArrayString nvarchar(4000), @ArrayDelimiter nchar(1),
@ArrayItemSelection smallint)
RETURNS nvarchar(4000)
AS
EXTERNAL NAME SQLArrayBuilder.SQLArrayBuilder.ChooseValueFromArray
GO
```

现在测试这个函数，第一个参数包含了3个逗号分隔的值。第二个参数指定了逗号作为分隔符，第三个值是要从数组中选择的值：

```
SELECT dbo.CountSalesOrderHeader
('Brian,Steve,Boris,Tony,Russ', ',', 3) Choice
```

这个查询返回：

```
Choice
Boris
```

这次从数组中选择第二个值：

```
SELECT dbo.CountSalesOrderHeader
('Brian,Steve,Boris,Tony,Russ', ',', 2) Choice
```

这个查询返回：

```
Choice
Steve
```

解析

这个技巧和CLR存储过程的示例的建立步骤一样。再说一次，程序集创建后需要编译。然后，使用CREATE ASSEMBLY把程序集加到数据库中。新建用户定义函数后，使用期望的3个输入参数以及合适的输出参数数据类型。UDF也引用了程序集、类以及函数名。

一点点来分析这段代码，你会看到我引入了一些在存储过程示例中也能看到的核心命名空间：

```
Imports System.Data
Imports System.Data.Sql
Imports System.Data.SqlTypes
Imports Microsoft.SqlServer.Server
```

还引入了正则表达式命名空间，这样就能使用Regex对象的功能了。命名空间是.NET提供的库类集合：

413

```
Imports System.Text.RegularExpressions
```

然后声明了我们的类名，在创建CLR函数的时候会引用它：

```
Public Class SQLArrayBuilder
```

声明的函数在圆括号中包含了3个输入参数，后面跟的是函数的返回数据类型（SqlString）：

```
Public Shared Function ChooseValueFromArray(ArrayString as String,
ArrayDelimiter as String, ArrayItemSelection as SqlInt16) as SqlString
```

然后，声明了一个新的字符串数组变量，为其填充由Regex.Split方法生成的数组。Regex.Split

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

方法用于根据由正则表达式匹配的位置（在这里就是分隔符）分割字符串：

```
Dim NewArrayString as String() =
Regex.Split(ArrayString, ArrayDelimiter)
```

VB.NET数组是从0开始的——也就是说数组中第一个值的索引为0，然后是1、2，等等。因为标量UDF的SQL Server调用方希望传递一个从1开始的数组选择值，所以我把传入的数组项选择减去1，这样它会从数组中选择合适的值：

```
Dim NewArrayItemSelection as SqlInt16=ArrayItemSelection-1
```

填充数组后，使用一个新的字符串变量来保存选择的值：

```
Dim ReturnString as SQLString = NewArrayString(NewArrayItemSelection)
```

这个值使用Return命令返回，之后就是关闭函数和类定义：

```
Return ReturnString

End Function

End Class
```

在这之后，编译程序集并使用CREATE ASSEMBLY将其导入数据库。然后使用CREATE FUNCTION创建函数来引用程序集、类和函数：

```
SQLArrayBuilder.SQLArrayBuilder.ChooseValueFromArray
```

测试这个函数，能正确解析逗号分隔的字符串并返回期望的标量值。

警告　本章中的示例旨在介绍CLR与SQL Server 2008集成的核心概念和功能。尽管传入合适值的时候，函数能正常工作，但是它没有包含错误捕捉代码来处理未预料的值。使用SAFE和EXTERNAL_ACCESS能限制可能损害SQL Server实例的行为，但是错误的输入值会导致给终端用户返回不友好的错误消息。在实际.NET代码中，请确保编写错误处理代码。

414

13.4.7　创建 CLR 触发器

在接下来的这个技巧中，会演示创建一个CLR触发器，它用来生成外部“控制文件”，这样就能用来通知一个假定的外部应用程序：处理已经结束了。

在这个示例的应用中，创建一个叫dbo.DataWarehouseLoadHistory的表。这个表在每天的数据仓库加载结束后插入一条记录。记录插入后，触发器会在外部的目录输出一个控制文件，通知遗留系统（假设这个系统不能以编程方式访问SQL Server 2008）。

首先，在用户定义的数据库内创建一个表：

```
USE BookStore
GO

CREATE TABLE dbo.DataWarehouseLoadHistory
(DataWarehouseLoadHistoryID int
   NOT NULL IDENTITY(1,1) PRIMARY KEY ,
LoadDT datetime NOT NULL)
GO
```

然后，使用vbc.exe编译下面的程序集代码：

```
Imports System
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Imports System.Data
Imports System.Data.Sql
Imports System.Data.SqlTypes
Imports System.Data.SqlClient
Imports Microsoft.SqlServer.Server
Imports System.IO


Public Class DW_Trigger

  Public Shared Sub ExportFile()

   Dim DWTrigger As SqlTriggerContext
   DWTrigger = SqlContext.TriggerContext

    If (DWTrigger.TriggerAction = _
     TriggerAction.Insert) Then

    Dim DWsw As StreamWriter = New _
     StreamWriter("C:\DataWarehouseLoadTrigger.txt")

    DWsw.WriteLine(Now())
    DWsw.Close()

    End If

   End Sub

End Class
```

在把程序集编译成DLL后，使用CREATE ASSEMBLY把它导入SQL Server：

415

```
CREATE ASSEMBLY DataWarehouseLoadNotification
FROM 'C:\Apress\Recipes\CLR\DataWarehouseLoadNotification.dll'
WITH PERMISSION_SET = EXTERNAL_ACCESS
GO
```

然后，创建触发器并映射到程序集子程序：

```
CREATE TRIGGER dbo.trg_i_DWNotify
ON dbo.DataWarehouseLoadHistory AFTER INSERT
AS
EXTERNAL NAME DataWarehouseLoadNotification.DW_Trigger.ExportFile
```

为了演示新的触发器，向DataWarehouseLoadHistory表插入一个新行：

```
INSERT dbo.DataWarehouseLoadHistory
(LoadDT)
VALUES(GETDATE())
```

INSERT子句触发了CLR触发器，然后在SQL Server实例所在机器的C:\盘下创建了一个通知文件（当然，在正式的情况下你应该把这个文件放在其他地方，让遗留系统能发现）。这个文件包含了触发器触发的当前日期和时间：

```
2/17/2008 2:43:47 PM
```

解析

这个技巧演示了创建CLR触发器，它对表的INSERT操作作出响应，创建一个文本文件。当然，对于每天接受很多新行的表来说，这个CLR数据库对象是不适合的（不间断地覆盖文件）。但在这个应

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用中，我假设数据只会定期更新，并且外部遗留应用程序会监控文件的任何修改。

创建这个CLR触发器的步骤和创建用户定义函数以及存储过程的步骤相似：编译新的程序集，载入SQL Server，然后使用CREATE TIGGER把它关联到数据库对象。

然而，需要指出的是，SqlTriggerContext类用于定义SQL Server中触发器的上下文信息：

```
Dim DWTrigger As SqlTriggerContext
DWTrigger = SqlContext.TriggerContext
```

一旦对象创建后，它就可以用来找到引起触发器触发的行动，或者检查哪些列被修改。在这个示例中，SqlTriggerContext对象用于检测触发器触发事件是不是INSERT操作，如果是的话就写入一个外部文件：

```
    If (DWTrigger.TriggerAction = _
     TriggerAction.Insert) Then
...
```

在编译DLL之后，创建了程序集并将其绑定到DataWarehouseLoadHistory表的触发器上。测试INSERT
416 子句，会在C:\盘下创建一个通知文件。

13.5　管理程序集

接下来的3个技巧会演示如何管理数据库程序集。我会演示如何查看程序集元数据、修改程序集权限以及从数据库中移除程序集。

13.5.1　查看程序集元数据

要查看当前数据库中的所有程序集，可以查询sys.assemblies系统目录视图。例如：

```
SELECT name, permission_set_desc
FROM sys.assemblies
```

这个查询返回：

```
name                                permission_set_desc
Microsoft.SqlServer.Types           UNSAFE_ACCESS
ReadFiles                           EXTERNAL_ACCESS
SQLArrayBuilder                     SAFE_ACCESS
DataWarehouseLoadNotification       EXTERNAL_ACCESS
```

解析

系统目录视图sys.assemblies能用于查看程序集名和被分配的安全级别。

13.5.2　修改程序集权限

可以使用ALTER ASSEMBLY命令（和CREATE ASSEMBLY的很多选项一样）来修改某个既有程序集权限的配置。

在这个示例中，把一个程序集的权限从EXTERNAL_ACCESS设置成SAFE：

```
ALTER ASSEMBLY ReadFiles
WITH PERMISSION_SET = SAFE
```

执行命令后，尝试执行关联这个程序集的存储过程：

```
EXEC dbo.usp_FileReader
N'C:\Program Files\Microsoft SQL Server\MSSQL10.AUGUSTUS\MSSQL\LOG\ERRORLOG.1'
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回下面的（部分）错误：

```
消息6522，级别16，状态1，过程usp_FileReader，第0行
在执行用户定义例程或聚合'usp_FileReader'期间出现.NET FrameWork错误。
System.Security.SecurityException: Request for the permission of type
'System.Security.Permissions.FileIOPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089' failed.
```

417

解析

尽管SQL Server可以修改程序集的权限级别，这个程序集的外部操作（从文件读取）不再被允许。这意味着写程序集的时候，必须想想需要哪个级别的权限。如果认为是SAFE的话，而它确实又需要访问外部资源，就可以使用ALTER ASSEMBLY来修改权限。

13.5.3 从数据库移除程序集

要从数据库移除程序集，需要使用DROP ASSEMBLY命令。部分语法如下：

```
DROP ASSEMBLY assembly_name [ ,...n ]
```

第一个参数是要从数据库中删除的一个程序集或者以逗号分隔的一列程序集的名字。例如：

```
DROP ASSEMBLY ReadFiles
```

解析

这个示例演示了删除一个程序集。任何存在的CLR对象引用（例如存储过程）都必须在移除程序集之前从数据库删除。如果不先删除引用的对象，会看到类似下面的错误消息：

```
消息6590，级别16，状态1，第1行
由于对象'usp_FileReader'引用了'ReadFiles'，DROP ASSEMBLY失败。
```

418

13

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 14 章 XML、分层和空间数据

在本章，我会通过一些技巧来讨论和演示XML与SQL Server的各种整合。也会介绍SQL Server 2008中新的原生hierarchyid、geometry和geography数据类型。

14.1 使用原生 XML

在SQL Server 2000中，如果要把XML数据保存在数据库中，就必须以字符或者二进制形式保存。如果仅仅使用SQL Server保存XML文档还不会太麻烦，但是如果你希望查询或修改保存在SQL Server中的文档就不那么简单了。SQL Server 2005新的原生xml数据类型能帮助解决这个问题。

关系型数据库设计者可能会关心这个新的数据类型，确实是这样。规格化的数据库具有性能和数据整合的优势，问题就来了，为什么我们需要一开始就保存XML文档？使用xml数据类型可以把关系型数据和非结构化数据保存在一起。微软不建议使用XML文档来运行高速应用程序。然而你可能会发现，有时候XML文档对结构化数据非常有用。例如，假设你的公司网站提供了一个在线合同。合同在网上开放，让所有的客户填写然后提交。提交的数据保存为xml数据类型。你会这么做可能是因为法律部门经常修改整个合同，增加一些字段并删除一些字段。不过，这些合同也只会一天填写几次，所以性能和吞吐量不是什么问题。另外一个使用原生xml数据类型的好理由是为了保存“状态”（state）。例如，如果你的.NET应用程序使用XML配置文件，就可以把它们保存在SQL Server数据库中以维护修改历史，并作为备份和恢复选项。

下面几个技巧会演示xml数据类型。

14.1.1 创建 XML 数据类型列

原生xml数据类型能用作表中的列、局部变量或参数的数据类型。保存在xml数据类型中的数据能包含XML文档或XML片段。XML片段是脱离一个顶层元素的XML实例。创建XML数据类型列与在表定义中使用它一样简单。例如，ChapterDESC列在下表中使用XML数据类型：


```
IF NOT EXISTS (SELECT name FROM sys.databases
               WHERE name = 'TestDB')
BEGIN
    CREATE DATABASE TestDB
END
GO

USE TestDB
GO

CREATE TABLE dbo.Book
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
(BookID int IDENTITY(1,1) PRIMARY KEY,
 ISBNNBR char(10) NOT NULL,
 BookNM varchar(250) NOT NULL,
 AuthorID int NOT NULL,
 ChapterDESC XML NULL)
GO
```

在第二个示例中，一个叫做@Book的局部变量被设置为xml数据类型，并且被赋为一个xml值（在下面的技巧中会演示如何使用这个值）：

```
DECLARE @Book XML

SET @Book =
CAST('<Book name="SQL Server 2000 Fast Answers">
<Chapters>
<Chapter id="1"> Installation, Upgrades... </Chapter>
<Chapter id="2"> Configuring SQL Server </Chapter>
<Chapter id="3"> Creating and Configuring Databases </Chapter>
<Chapter id="4"> SQL Server Agent and SQL Logs </Chapter>
</Chapters>
</Book>' as XML)
```

在第三个示例中，在存储过程中使用一个xml数据类型作为输入参数：

```
CREATE PROCEDURE dbo.usp_INS_Book
   @ISBNNBR char(10),
   @BookNM varchar(250),
   @AuthorID int,
   @ChapterDESC xml
AS

INSERT dbo.Book
(ISBNNBR, BookNM, AuthorID, ChapterDESC)
VALUES (@ISBNNBR, @BookNM, @AuthorID, @ChapterDESC)

GO
```

解析

这个技巧演示了如何在表的列定义中、局部变量以及存储过程的输入参数中使用xml数据类型。语法和其他SQL Server数据类型的使用没有什么不同。下个技巧演示如何使用T-SQL将XML数据插入到表中。

420

14.1.2 插入 XML 数据到列

在这个技巧中，会演示将一个XML文档插入到前面技巧创建的表中。我们使用了INSERT命令，XML文档包含在单引号中（像一个字符串那样），然后使用CAST将其显式转化为xml数据类型：

```
INSERT dbo.Book
(ISBNNBR, BookNM, AuthorID, ChapterDESC)
VALUES ('570X000000',
      'SQL Server 2008 T-SQL Recipes',
      55,
CAST('<Book name="SQL Server 2008 T-SQL Recipes">
<Chapters>
<Chapter id="1"> SELECT </Chapter>
<Chapter id="2"> INSERT,UPDATE,DELETE </Chapter>
<Chapter id="3"> Transactions, Locking, Blocking, and Deadlocking </Chapter>
<Chapter id="4"> Tables </Chapter>
<Chapter id="5"> Indexes </Chapter>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<Chapter id="6"> Full-text search </Chapter>
</Chapters>
</Book>' as XML))
```

这个查询返回：

```
(1行受影响)
```

在第二个示例中，有一个叫@Book的局部变量被设置为xml数据类型并且赋了一个xml值。然后在表的INSERT操作中使用它：

```
DECLARE @Book XML

SET @Book =
CAST('<Book name="SQL Server 2000 Fast Answers">
<Chapters>
<Chapter id="1"> Installation, Upgrades... </Chapter>
<Chapter id="2"> Configuring SQL Server </Chapter>
<Chapter id="3"> Creating and Configuring Databases </Chapter>
<Chapter id="4"> SQL Server Agent and SQL Logs </Chapter>
</Chapters>
</Book>' as XML)

INSERT dbo.Book
(ISBNNBR, BookNM, AuthorID, ChapterDESC)
VALUES ('1590591615',
      'SQL Server 2000 Fast Answers',
      55,
      @Book)
```

这个查询返回：

```
(1行受影响)
```

421

解析

在两个INSERT示例中，都使用了CAST函数把ChapterDESC列的XML数据显式转化为xml，然后SQL Server会检测以确保它是合式的（在这里，合式的指的是它符合了XML文档的一般规则）。例如，如果文档片段丢失了结束元素</Book>，则会产生下面的错误：

```
消息9400，级别16，状态1，第1行
XML分析：行9，字符12，意外的输入结尾
```

然而，在本例中定义的xml列是无类型的（untyped）。如果xml列是无类型的，那么插入到列的数据也不会使用XML架构进行验证。XML架构用于为XML文档定义可用的元素和属性。下个技巧会讨论。

14.1.3　使用架构验证 XML 数据

XML架构（或者叫做XML架构定义或XSD）定义了XML文档的元素、属性、数据类型和允许的值。使用CREATE XML SCHEMA COLLECTION，你可以为SQL Server增加XML架构定义然后使用它们来约束XML数据类型列、局部变量或参数。

提示　要学习XML架构基本原理，可以访问万维网协会（W3C）标准网站：http://www.w3.org/TR/XMLschema-0/。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

CREATE XML SCHEMA COLLECTION命令用于新增XML架构，语法如下：

```
CREATE XML SCHEMA COLLECTION [ <relational_schema>. ]sql_identifier
AS Expression
```

这个命令接受两个参数，第一个是新的XML架构的唯一名字，而第二个参数就是XML架构的主体。要增加其他XML架构到既有集合中，可以使用ALTER XML SCHEMA COLLECTION。语法如下：

```
ALTER XML SCHEMA COLLECTION [ relational_schema. ]sql_identifier
ADD 'Schema Component'
```

要从数据库中移除整个XML架构集合，可以使用DROP XML SCHEMA命令，语法如下：

```
DROP XML SCHEMA COLLECTION [ relational_schema. ]sql_identifier
```

删除一个既有XML架构集合的唯一参数就是集合名。

在本例中，创建了一个叫BookStoreCollection的新XML架构集合，在其中包含了一个XML架构的定义：

422

```
CREATE XML SCHEMA COLLECTION BookStoreCollection
AS
N'<xsd:schema targetNamespace="http://JOEPROD/BookStore"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:sqltypes="http://schemas.microsoft.com/sqlserver/2004/sqltypes"
elementFormDefault="qualified">
<xsd:import namespace=
"http://schemas.microsoft.com/sqlserver/2004/sqltypes" />
<xsd:element name="Book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="BookName" minOccurs="0">
          <xsd:simpleType>
            <xsd:restriction base="sqltypes:varchar">
              <xsd:maxLength value="50" />
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
<xsd:element name="ChapterID" type="sqltypes:int"
minOccurs="0" />
        <xsd:element name="ChapterNM" minOccurs="0">
          <xsd:simpleType>
            <xsd:restriction base="sqltypes:varchar">
              <xsd:maxLength value="50" />
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>'
```

这个查询返回：

```
命令成功执行。
```

一旦创建，就可以使用系统目录视图sys.XML_schema_collections和sys.XML_schema_namespaces来验证一个XML架构的存在。第一个查询显示了定义在数据库中的所有架构集合：

```
SELECT name
FROM sys.XML_schema_collections
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ORDER BY create_date
```

这个查询返回：

```
name
sys
BookStoreCollection
```

第二个查询显示了某个XML架构集合中的命名空间（命名空间唯一标识了元素和属性的范围，有
423 助于唯一标识这些组件）：

```
SELECT n.name
FROM sys.XML_schema_namespaces n
INNER JOIN sys.XML_schema_collections c ON
   c.XML_collection_id = n.XML_collection_id
WHERE c.name = 'BookStoreCollection'
```

这个查询返回：

```
name
http://JOEPROD/BookStore
```

一旦某个架构集合可用，就可以把它绑定到表中的xml列上，可以通过在数据类型定义后的圆括号内引用它来实现。例如，ChapterDESC列绑定到BookStoreCollection XML架构集合上：

```
CREATE TABLE dbo.BookInfoExport
(BookID int IDENTITY(1,1) PRIMARY KEY,
 ISBNNBR char(10) NOT NULL,
 BookNM varchar(250) NOT NULL,
 AuthorID int NOT NULL,
 ChapterDESC xml (BookStoreCollection) NULL)
```

这个xml列现在只允许带类型的xml值（遵从定义的XML架构集合的XML文档）。尝试指定一个不遵从列定义的XSD的XML值会引发一个错误（例如如果期望的元素或属性丢失）。使用架构集合引用的DOCUMENT或CONTENT关键字决定XML是否只允许完整的XML文档（DOCUMENT）或XML片段（CONTENT）。

例如，下面的局部变量需要一个遵从XML架构集合的完整XML文档：

```
DECLARE @Book XML (DOCUMENT BookStoreCollection)
```

解析

这个技巧快速介绍了SQL Server内建的XML架构功能。使用XML架构集合，可以验证和约束xml数据类型中XML数据的内容。对于一般的XML结构，无类型的XML数据仍然要进行验证，但是通过使用XML架构集合能应用更加复杂的验证和约束。

提示　现在SQL Server 2008允许对通配符进行宽松的验证。SQL Server 2005支持strict和skip。strict值表示对所有内容进行全面的验证，skip指定不对内容进行验证。新的lax值指使SQL Server验证在架构中定义的属性和元素，但忽略对未定义属性和元素的验证。

14.1.4　获取 XML 数据

xml数据类型列能使用XQuery方法来查询。XQuery是用于搜索XML文档的查询语言。表14-1中描
424 述的这些XQuery方法整合在SQL Server中并且能在一般T-SQL查询中使用。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表14-1 XQuery方法

方 法	描 述
exist	如果XQuery表达式计算为TRUE则返回1，计算为FALSE时返回0
modify	对XML数据执行更新操作（在这个技巧之后演示）
nodes	把XML数据拆分成关系型数据，进行节点到行的映射
query	根据一个XQuery表达式返回XML结果
value	根据一个XQuery表达式返回标量SQL数据类型

提示 要深入了解XQuery基础知识，可访问万维网协会（W3C）标准网站http://www.w3.org/TR/xquery。XQuery支持使用for、let、where、order by和return子句的迭代语法（首字母缩写FLWOR）。在SQL Server 2005中，不支持let子句。现在在SQL Server 2008中支持了。

要演示这些方法，我们创建一个有xml数据类型列的新表，并且插入3行：

```
CREATE TABLE dbo.BookInvoice
(BookInvoiceID int IDENTITY(1,1) PRIMARY KEY,
 BookInvoiceXML XML NOT NULL)
GO

INSERT dbo.BookInvoice
(BookInvoiceXML)
VALUES ('<BookInvoice invoicenumber="1" customerid="22" orderdate="7/1/2008">
<OrderItems>
<Item id="22" qty="1" name="SQL Fun in the Sun"/>
<Item id="24" qty="1" name="T-SQL Crossword Puzzles"/>
</OrderItems>
</BookInvoice>')

INSERT dbo.BookInvoice
(BookInvoiceXML)
VALUES ('<BookInvoice invoicenumber="1" customerid="40" orderdate="7/11/2008">
<OrderItems>
<Item id="11" qty="1" name="MCDBA Cliff Notes"/>
</OrderItems>
</BookInvoice>')

INSERT dbo.BookInvoice
(BookInvoiceXML)
VALUES ('<BookInvoice invoicenumber="1" customerid="9" orderdate="7/22/2008">
<OrderItems>
<Item id="11" qty="1" name="MCDBA Cliff Notes"/>
<Item id="24" qty="1" name="T-SQL Crossword Puzzles"/>
</OrderItems>
</BookInvoice>')
```

在第一个示例中，使用exist方法查找表中ID为11的已购Item的所有行。

```
SELECT BookInvoiceID
FROM dbo.BookInvoice
WHERE BookInvoiceXML.exist
('/BookInvoice/OrderItems/Item[@id=11]') = 1
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
BookInvoiceID
2
3
```

下面的示例演示了nodes方法，它把文档拆分为关系型行集。使用一个局部变量来从BookInvoice表填充一个XML文档，然后使用nodes方法引用它。这个查询检索文档并列出每个BookInvoice/OrderItems/Item节点的ID元素：

```
DECLARE @BookInvoiceXML xml

SELECT @BookInvoiceXML = BookInvoiceXML
FROM dbo.BookInvoice
WHERE BookInvoiceID = 2

SELECT BookID.value('@id','integer') BookID
FROM @BookInvoiceXML.nodes('/BookInvoice/OrderItems/Item')
AS BookTable(BookID)
```

前面一个查询返回虚拟表BookTable中Item的ID值：

```
BookID
11
```

下面的示例演示了query方法，它用来返回某个XML文档的两个Item元素：

```
DECLARE @BookInvoiceXML XML

SELECT @BookInvoiceXML = BookInvoiceXML
FROM dbo.BookInvoice
WHERE BookInvoiceID = 3

SELECT @BookInvoiceXML.query('/BookInvoice/OrderItems')
```

这个查询返回：

```
<OrderItems>
  <Item id="11" qty="1" name="MCDBA Cliff Notes" />
  <Item id="24" qty="1" name="T-SQL Crossword Puzzles" />
</OrderItems>
```

本技巧的最后一个示例演示了value方法，它用于找到BookInvoiceXML xml列中第一个和第二个Item中不同的图书名：

```
SELECT DISTINCT
BookInvoiceXML.value
('(/BookInvoice/OrderItems/Item/@name)[1]', 'varchar(30)') as BookTitles
FROM dbo.BookInvoice
UNION
SELECT DISTINCT
BookInvoiceXML.value
('(/BookInvoice/OrderItems/Item/@name)[2]', 'varchar(30)')
FROM dbo.BookInvoice
```

426

使用UNION来合并两个结果集，/BookInvoice/OrderItems/Item节点的两个层次使用两个独立的查询进行剖析（NULL值来自只保存了一个Item的XML片段）：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
BookTitles
NULL
MCDBA Cliff Notes
SQL Fun in the Sun
T-SQL Crossword Puzzles
```

解析

XQuery方法让我们能查询和修改xml数据类型中的数据（在本章稍后会有演示修改）。本技巧中大多数的示例使用XMLColumn.MethodName之类的格式。

例如，用在BookInvoiceXML xml列的exist方法显示了ID为11的Item。XQuery表达式跟在方法名之后的圆括号中：

```
BookInvoiceXML.exist ('/BookInvoice/OrderItems/Item[@id=11]') = 1
```

nodes函数示例包含了定义以碎片格式返回结果的XQuery表达式，随后是圆括号内的新结果表名和列名：

```
@BookInvoiceXML.nodes('/BookInvoice/OrderItems/Item')
AS BookTable(BookID)
```

query方法使用了一个简单的XQuery表达式来返回结果中的Item元素：

```
@BookInvoiceXML.query('/BookInvoice/OrderItems/Item')
```

包含XQuery表达式的value方法为每行返回标量值，由第二个参数定义了数据类型：

```
BookInvoiceXML.value
('(/BookInvoice/OrderItems/Item/@name)[2]', 'varchar(30)')
```

14.1.5 修改 XML 数据

xml数据类型列能使用modify方法和UPDATE联合来修改，使用它可以插入、更新或删除xml数据类型列中的XML节点。

本示例演示了使用modify方法来将新的Item插入到既有的XML文档（特别地，将一个新的Item插入到/BookInvoice/OrderItems节点）：

```
UPDATE dbo.BookInvoice
SET BookInvoiceXML.modify
('insert <Item id="920" qty="1" name="SQL Server 2008 Transact-SQL Recipes"/>
into (/BookInvoice/OrderItems)[1]')
WHERE BookInvoiceID = 2
```

检查BookInvoice XML文档的这行，确认增加了新的Item：

```
<BookInvoice invoicenumber="1" customerid="40" orderdate="7/11/2008">
  <OrderItems>
    <Item id="11" qty="1" name="MCDBA Cliff Notes" />
    <Item id="920" qty="1" name="SQL Server 2008 Transact-SQL Recipes" />
  </OrderItems>
</BookInvoice>
```

解析

modify函数也使用了XMLColumn.MethodName格式，并且在圆括号中使用了XQuery插入表达式来插入新的Item到既有文档：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
BookInvoiceXML.modify
('insert <Item id="920" qty="1" name="SQL Server 2008 T-SQL Recipes"/>
into (/BookInvoice/OrderItems)[1]')
```

用于插入新的Item元素的insert命令是XQuery语言的扩展，称作XML DML。其他XML DML命令包括用于更新XML数据的replace语句以及用于从XML文档或片段移除节点的delete语句。

14.1.6　XML索引

可以使用XML索引提高XML数据类型列的查询性能。要创建XML索引，表必须已经在主键上定义了聚集索引。

XML列只能定义一个主XML索引，以及最多3个辅助索引（具有下面描述的不同类型）。使用CREATE INDEX命令定义XML索引。部分语法如下：

```
CREATE [ PRIMARY ] XML INDEX index_name
    ON <object> ( xml_column_name )
     [ USING XML INDEX xml_index_name
         [ FOR { VALUE | PATH | PROPERTY } ] ]
     [ WITH ( <xml_index_option> [ ,...n ] ) ][ ; ]
```

为XML列创建索引要使用几个和一般表索引一样的参数（更多信息请参阅第5章）。表14-2描述了这个命令的XML相关参数。

表14-2　CREATE XML INDEX参数

参　数	描　述
object	这个参数指定了要添加的索引所属的表名
XML_column_name	这个参数指定了XML数据类型列名
XML_index_name	XML索引的唯一名
VALUE \| PATH \| PROPERTY	这些参数仅针对辅助索引并且和XQuery优化相关。VALUE辅助索引用于根据模糊路径创建索引。PATH辅助索引用来根据路径和节点值创建索引。PROPERTY辅助索引用来根据某个路径查询节点值来创建索引

428

在第一个示例中，在一个xml数据类型列上创建了主XML索引：

```
CREATE PRIMARY XML INDEX idx_XML_Primary_Book_ChapterDESC
    ON dbo.Book(ChapterDESC)
GO
```

然后，在同样的xml列上使用另外一个名字创建VALUE辅助索引。在辅助索引中增加USING子句，在FOR子句中指定xml数据类型列除了既有的主索引外还另外增加一个VALUE索引：

```
CREATE XML INDEX idx_XML_Value_Book_ChapterDESC
    ON dbo.Book(ChapterDESC)
   USING XML INDEX idx_XML_Primary_Book_ChapterDESC
FOR VALUE
GO
```

可以使用sys.XML_indexes系统目录视图来查看数据库中使用的XML索引。在这个查询中，列出了某个表上所有的XML索引：

```
SELECT  name, secondary_type_desc
FROM    sys.XML_indexes
WHERE object_id = OBJECT_ID('dbo.Book')
```

这个查询返回XML索引的名字，如果索引是辅助索引，则还会返回其类型：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
name                                secondary_type_desc
idx_XML_Primary_Book_ChapterDESC    NULL
idx_XML_Value_Book_ChapterDESC      VALUE
```

XML索引一旦创建就能像一般索引那样使用ALTER INDEX和DROP INDEX命令来修改或者删除。

解析

由于XML文档能在一列和一行内存储多达2GB的数据，因此当你尝试查询存储在XML列中数据的时候查询性能会很差。如果你计划频繁查询XML数据类型的数据，请务必使用XML索引。索引xml数据类型会在内部以列表形式保存XML数据，能让我们更有效地查询层次化数据。

XML索引一开始看起来有点多余，因为你要为相同的xml数据类型列增加辅助索引。根据希望执行的不同类型的XQuery查询增加不同类型的辅助索引有助于提升性能。总共可以在一个xml数据类型列上最多使用4个索引：一个主索引和3个辅助索引。主XML索引必须在创建辅助索引之前创建。PATH辅助索引用于为使用XQuery指定xml列的路径和节点值的查询提升性能。PROPERTY辅助索引用于为使用XQuery指定某个路径来获取节点值的查询提升性能。VALUE辅助索引用于为使用模糊路径获取数据的查询提升性能（例如，不管文档中的节点是否存在，都可以使用带有//的XPath表达式来查找它们）。 429

14.2 在XML文档与关系型数据之间进行转换

在下面的技巧中，会演示如何使用FOR XML把关系型数据集转化为分层XML格式。在这之后，会演示如何使用OPENXML把XML格式转化为关系型数据集。

14.2.1 将关系型数据格式化为XML

SQL Server 2000中就引入了FOR XML，它通过以XML格式返回关系查询结果来扩展SELECT语句。FOR XML操作有4种不同的模式：RAW、AUTO、EXPLICIT和PATH。

对于RAW模式，结果集中的每一行都生成一行元素，结果集中的每一列转化为元素的属性。

在这个示例中，使用FOR XML RAW以XML格式返回HumanResources.Shift表的结果。使用TYPE选项以XML数据类型返回结果，使用ROOT定义将被嵌套的结果所属的顶层元素：

```
USE AdventureWorks
GO

SELECT ShiftID, Name
FROM HumanResources.Shift
FOR XML RAW('Shift'), ROOT('Shifts'), TYPE
```

这个查询返回：

```
<Shifts>
   <Shift ShiftID="1" Name="Day" />
   <Shift ShiftID="2" Name="Evening" />
   <Shift ShiftID="3" Name="Night" />
</Shifts>
```

FOR XML AUTO模式为SELECT语句的结果集创建XML元素，并且根据SELECT子句的列自动嵌套数据。AUTO和RAW有相同的选项。

在本例中，我们查询AdventureWorks数据库中的Employee、Shift 和Department信息——XML AUTO

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

自动处理结果的层次：

```
SELECT TOP 3 BusinessEntityID,
      Shift.Name,
      Department.Name
FROM  HumanResources.EmployeeDepartmentHistory Employee
INNER JOIN HumanResources.Shift Shift ON
   Employee.ShiftID = Shift.ShiftID
INNER JOIN HumanResources.Department Department ON
   Employee.DepartmentID = Department.DepartmentID
ORDER BY BusinessEntityID
FOR XML AUTO, TYPE
```

430

这个查询返回：

```
<Employee BusinessEntityID="1">
  <Shift Name="Day">
    <Department Name="Executive" />
  </Shift>
</Employee>
<Employee BusinessEntityID="2">
  <Shift Name="Day">
    <Department Name="Engineering" />
  </Shift>
</Employee>
<Employee BusinessEntityID="3">
  <Shift Name="Day">
    <Department Name="Engineering" />
  </Shift>
</Employee>
```

注意到，第二个INNER JOIN让Department的值成为Shift表值的子级。然后Shift元素就成为了Employee元素的子元素。不过，重新调整SELECT子句中列的次序会影响返回的层次关系，例如：

```
SELECT TOP 3
      Shift.Name,
      Department.Name,
      BusinessEntityID
FROM  HumanResources.EmployeeDepartmentHistory Employee
INNER JOIN HumanResources.Shift Shift ON
   Employee.ShiftID = Shift.ShiftID
INNER JOIN HumanResources.Department Department ON
   Employee.DepartmentID = Department.DepartmentID
ORDER BY Shift.Name, Department.Name, BusinessEntityID
FOR XML AUTO, TYPE
```

这次层次的顶端是Shift，子元素是Department，且Employee是Department的子元素：

```
<Shift Name="Day">
  <Department Name="Document Control">
    <Employee BusinessEntityID="217" />
    <Employee BusinessEntityID="219" />
    <Employee BusinessEntityID="220" />
  </Department>
</Shift>
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使用FOR XML EXPLICIT模式可以更多地控制XML结果，可以定义列作为元素还是属性。EXPLICIT参数和RAW以及AUTO有相同的意义和使用方法；不过，EXPLICIT还使用指示（directive）来定义结果元素和属性。例如，下面的查询把VenderID和CreditRating列作为属性，VendorName列作为元素。使用元素名、标签号、属性和指示在列别名之后定义列：

431

```
SELECT TOP 3
   1 AS Tag,
   NULL AS Parent,
   BusinessEntityID AS [Vendor!1!VendorID],
   Name AS [Vendor!1!VendorName!ELEMENT],
   CreditRating AS [Vendor!1!CreditRating]
FROM Purchasing.Vendor
ORDER BY CreditRating
FOR XML EXPLICIT, TYPE
```

这个查询返回：

```
<Vendor VendorID="1496" CreditRating="1">
  <VendorName>Advanced Bicycles</VendorName>
</Vendor>
<Vendor VendorID="1492" CreditRating="1">
  <VendorName>Australia Bike Retailer</VendorName>
</Vendor>
<Vendor VendorID="1500" CreditRating="1">
  <VendorName>Morgan Bike Accessories</VendorName>
</Vendor>
```

在EXPLICIT模式下需要Tag列来生成XML文档输出，每个标签号表示一个结构化的元素。也需要Parent列别名，它提供了所有父元素的层次信息。Parent列引用了父元素的标签。如果Parent列为NULL，就代表没有父元素，并且为顶层级别。

前面查询的FOR XML子句中的TYPE指示用于返回真正的SQL Server原生xml数据类型的结果，允许我们以XML形式存储结果或使用XQuery查询它。

接着，FOR XML PATH选项定义了列名和别名作为XPath表达式。在本章开始的时候提到过，XPath是用于在XML文档中搜索数据的语言。

提示 更多有关XPath的信息，请访问万维网协会（W3C）标准网站 http://www.w3.org/TR/xpath。

FOR XML PATH使用和其他FOR XML形式一样的一些参数和关键字。然而，不同的是，在SELECT子句中，使用XPath语法定义元素、子元素、属性和数据值。

例如：

```
SELECT Name as "@Territory",
      CountryRegionCode as "@Region",
      SalesYTD
FROM Sales.SalesTerritory
WHERE SalesYTD > 6000000
ORDER BY SalesYTD DESC
FOR XML PATH('TerritorySales'), ROOT('CompanySales'), TYPE
```

这个查询返回：

14

432

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<CompanySales>
   <TerritorySales Territory="Southwest" Region="US">
      <SalesYTD>8351296.7411</SalesYTD>
   </TerritorySales>
   <TerritorySales Territory="Canada" Region="CA">
      <SalesYTD>6917270.8842</SalesYTD>
   </TerritorySales>
</CompanySales>
```

这个查询返回的结果中根元素节点是CompanySales，下面有子元素TerritorySales。TerritorySales元素然后被标记Territory和Region代码（在SELECT子句中都以@符号开头）。SaleYTD没有标记Xpath指示，因此也就成为了TerritorySales的子元素。

解析

FOR XML命令包含在SELECT查询的最后以返回XML格式的数据。AUTO和RAW模式提供了快速的、半自动格式化结果的方法，而EXPLICIT和PATH提供了对元素和属性分配以及层次数据的更多控制。但是，对于那些对Xpath语言更熟悉的开发者来说，FOR XML PATH比EXPLICIT更简单。

本技巧中演示的FOR XML选项是从结果集中创建XML的最常见的方法。使用FOR XML生成XML文档片段比使用T-SQL手动方法创建层次关系更简单。但是记住，还能够以编程方式创建XML文档（例如使用.NET）。

14.2.2　将 XML 转换为关系型

FOR XML是把关系型查询结果转化为XML格式，而OPENXML把XML格式转化为关系型形式。要实现这种转换，可以使用sp_XML_preparedocument系统存储过程来创建指向XML文档的内部指针，然后使用OPENXML来返回行集数据。

OPENXML命令的语法如下：

```
OPENXML(idoc ,rowpattern, flags)
[WITH (SchemaDeclaration | TableName)]
```

表14-3描述了此命令的参数。

表14-3　OPENXML参数

参　数	描　述
idoc	由sp_XML_preparedocument系统存储过程生成的XML文档的内部表示
rowpattern	这个参数定义了用于从XML文档返回节点的XPath模式
flags	如果标志位使用0，结果默认为以属性为中心的映射。如果标志位使用1，首先应用以属性为中心的映射，然后对没有处理的列使用以元素为中心的映射。标志位2使用以元素为中心的映射。标志位8指定不应将已使用的数据复制到溢出属性
SchemaDeclaration\|TableName	SchemaDeclaration定义了输出的列名（行集名）、列类型（有效的数据类型）、列模式（可选的XPath模式），以及可选的元数据属性（有关XML节点）。如果使用Tablename，那么保存行集数据的表必须已经存在

433

在这个示例中，我们把XML文档保存在一个局部变量中，然后传入存储过程，它使用OPENXML来将其转化为关系型行集。首先，创建存储过程：

```
CREATE PROCEDURE dbo.usp_SEL_BookXML_Convert_To_Relational
   @XMLDoc xml
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
AS

DECLARE @docpointer int

EXEC sp_XML_preparedocument @docpointer OUTPUT, @XMLdoc

SELECT    Chapter, ChapterNM
FROM       OPENXML (@docpointer, '/Book/Chapters/Chapter',0)
            WITH (Chapter int '@id',
              ChapterNM varchar(50) '@name' )
GO
```

然后，为一个局部xml数据类型变量赋值并传入新的存储过程：

```
DECLARE @XMLdoc XML
SET @XMLdoc =
'<Book name="SQL Server 2000 Fast Answers">
   <Chapters>
      <Chapter id="1" name="Installation, Upgrades"/>
      <Chapter id="2" name="Configuring SQL Server"/>
      <Chapter id="3" name="Creating and Configuring Databases"/>
      <Chapter id="4" name="SQL Server Agent and SQL Logs"/>
   </Chapters>
   </Book>'
EXEC dbo.usp_SEL_BookXML_Convert_To_Relational @XMLdoc
```

这个查询返回：

```
Chapter     ChapterNM
1           Installation, Upgrades
2           Configuring SQL Server
3           Creating and Configuring Databases
4           SQL Server Agent and SQL Logs
```

434

解析

这个示例从创建一个把XML文档片段转化为关系型数据集的存储过程开始。存储过程有一个定义为xml数据类型的输入参数：

```
CREATE PROCEDURE dbo.usp_SEL_BookXML_Convert_To_Relational
   @XMLDoc xml
AS
```

声明一个作为sp_XML_preparedocument系统存储过程输出参数的局部变量来保存内部文档指针的值：

```
DECLARE @docpointer int
```

然后，调用一个带OUTPUT参数的系统存储过程，它的第二个参数就是输入的xml数据类型参数：

```
EXEC sp_XML_preparedocument @docpointer OUTPUT, @XMLdoc
```

接下来，SELECT语句在FROM子句中引用OPENXML函数，并且指定结果中返回的两列的名字：

```
SELECT     Chapter, ChapterNM
FROM        OPENXML
```

OPENXML命令的第一个参数是内部指针变量。第二个参数是要在XML文档中使用的节点的XPATH表达式。第三个参数是标志位，它指定了一个以属性为中心的映射：

```
(@docpointer, '/Book/Chapters/Chapter',0)
```

WITH子句定义了实际结果的输出。在这里定义了两列，一个是Chapter另外一个是ChapterNM。@id

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

指定了ID属性映射到Chapter列，而@name属性映射到ChapterNM列：

```
WITH (Chapter int '@id',
  ChapterNM varchar(50) '@name' )
```

创建存储过程之后，对局部变量填充XML片段，然后传给存储过程，返回两列和4行。

14.3　使用原生分层数据

SQL Server 2008引入了新的hierarchyid数据类型，可以用它来做本地存储并且在树层次结构中管理其位置。使用这个新数据类型可以简洁地表示层次结构中节点的位置，与xml数据类型相似，hierarchyid包含几个可以用来操作或遍历层次结构的内置方法。这个新的提升使得存储和查询分层数据更为容易，而不需要创建自己的方法。

随后几个技巧将演示如何使用hierarchyid数据类型来存储、操作和查询分层数据。

14.3.1　存储分层数据

这个技巧演示了存储网页布局的层次结构。在这个示例中，希望表示一个根网页，以及两级页面
435 和相关的同级节点。

先创建用来保存网页层次结构的表。表定义的第一列WebpageLayoutID将包含hierarchyid数据类型数据。第二列PositionDESC，是使用hierarchyid数据类型列的GetLevel方法的计算列，显示了指定行在树中的位置。最后一列PageURL，只是包含实际网页的URL。

提示　hierarchyid数据类型包含几个可以用来操作和遍历节点值的方法。在随后的技巧中会更详细地解释和演示这些内容。

```
USE TestDB
GO

CREATE TABLE dbo.WebpageLayout
   (WebpageLayoutID hierarchyid NOT NULL,
    PositionDESC as WebpageLayoutID.GetLevel(),
    PageURL nvarchar(50) NOT NULL)
GO
```

接着，将插入表示网站层次结构的根的新行：

```
INSERT dbo.WebpageLayout
(WebpageLayoutID, PageURL)
VALUES
('/', 'http://joesack.com')
```

注意字符串中用/来表示层次的根。这会自动将插入转换为原生hierarchyid格式（二进制格式）。下一步，将查询数据查看它是如何存储的：

```
SELECT WebpageLayoutID, PositionDESC, PageURL
FROM dbo.WebpageLayout
```

这个查询返回：

```
WebpageLayoutID PositionDESC PageURL
0x 0 http://joesack.com
```

注意最初的/值被重新定义为0x十六进制值。使用斜杠字符来表示层次路径。一个斜杠用来表示

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

树根。用被斜杠分隔的整数值来组成连续的级别。

接着，将插入两个表示根网页子节点的新行，这两个网页处于相同的级别（因此它们是同级）：

```
INSERT dbo.WebpageLayout
(WebpageLayoutID, PageURL)
VALUES
('/1/', 'http://joesack.com/WordPress/')

INSERT dbo.WebpageLayout
(WebpageLayoutID, PageURL)
VALUES
('/2/', 'http://joesack.com/SQLFastTOC.htm')
```

436

再次查询目前为止的结果：

```
SELECT WebpageLayoutID, PositionDESC, PageURL
FROM dbo.WebpageLayout
```

这个查询返回：

```
WebpageLayoutID  PositionDESC  PageURL
0x               0             http://joesack.com
0x58             1             http://joesack.com/WordPress/
0x68             1             http://joesack.com/SQLFastTOC.htm
```

注意PositionDESC显示了位置为0的根网页，和两个位置为1的子节点。

接下来，将演示添加两个http://joesack.com/WordPress/子节点的新网页。与前面示例不同的是，将使用hierarchyid数据类型的GetDescendant方法来填充WebpageLayoutID：

```
DECLARE @ParentWebpageLayoutID hierarchyid
SELECT @ParentWebpageLayoutID = CONVERT(hierarchyid, '/1/')

INSERT dbo.WebpageLayout
(WebpageLayoutID, PageURL)
VALUES
(@ParentWebpageLayoutID.GetDescendant(NULL,NULL),
'http://joesack.com/WordPress/?page_id=2')

INSERT dbo.WebpageLayout
(WebpageLayoutID, PageURL)
VALUES
(@ParentWebpageLayoutID.GetDescendant(NULL,NULL),
'http://joesack.com/WordPress/?page_id=9')
```

这个查询返回：

```
(1行受影响)
(1行受影响)
```

随后的查询使用ToString方法来显示hierarchyid数据类型的字符串表示，而不是显示WebpageLayoutID的原生格式：

```
SELECT WebpageLayoutID.ToString() as WebpageLayoutID, PositionDESC, PageURL
FROM dbo.WebpageLayout
```

注意，使用GetDescendant方法将新插入的行放置在适当的父节点下面，而不是将值硬编码写入到INSERT中：

```
WebpageLayoutID  PositionDESC  PageURL
/                0             http://joesack.com
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
/1/                1          http://joesack.com/WordPress/
/1/1/              2          http://joesack.com/WordPress/?page_id=2
/1/1/              2          http://joesack.com/WordPress/?page_id=9
/2/                1          http://joesack.com/SQLFastTOC.htm
```

437

解析

这个技巧介绍了SQL Server 2008的hierarchyid数据类型，并通过演示如何创建使用hierarchyid数据类型的表开始。在这之后，为根网页插入新行。使用斜杠来表示层次路径。一个斜杠表示树根。使用被斜杠分隔的整数来组成连续的级别：

```
/
/1/
/1/1/
/2/
/2/1/
/2/2/
```

提示　在同一级别的节点排序也可以使用小数点来指定。例如，/1/1.5/。

在插入根节点之后，查询表显示已经将斜杠转换为十六进制值。然后插入两个新页面，它们是根节点的子节点。随后又为其中一个子节点创建两个新子节点，但没有手工指定节点路径，而是使用GetDescendant函数来指定了路径。在插入之后查询表，查询结果会显示被使用的正确位置和路径。

14.3.2　返回指定的祖先

前面的技巧介绍了如何使用原生hierarchyid数据类型。hierarchyid数据类型包含了可以用来检索和遍历数据值的方法集。我会在随后的几个技巧中通过示例一步步地讲解如何使用每个方法。

使用GetAncestor方法可以通过从指定位置提升一些级别数来返回层次结构中的某个位置。输入参数定义了要提升的级别数。在这个示例中，创建了两个局部变量来保存原始节点值和上一级的祖先值：

```
DECLARE @WebpageLayoutID hierarchyid
DECLARE @New_WebpageLayoutID hierarchyid
SELECT @WebpageLayoutID = CONVERT(hierarchyid, '/1/1/')

SELECT @New_WebpageLayoutID = @WebpageLayoutID.GetAncestor( 1 )
SELECT @New_WebpageLayoutID.ToString()
```

这个查询返回1/1节点的父节点，即/1/：

```
/1/
```

解析

这个示例演示了如何使用hierarchyid数据类型的各种可用方法。在包含数据值的局部变量后使用点号来引用这些方法，与在本章前面演示的xml数据类型使用方法相似（例如，modify方法）。

438

在使用方法之前首先要声明hierarchyid变量：

```
DECLARE @WebpageLayoutID hierarchyid
```

定义另一个变量来保存新的hierarchyid值：

```
DECLARE @New_WebpageLayoutID hierarchyid
```

在这之后，使用SELECT来给变量赋值：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SELECT @WebpageLayoutID = CONVERT(hierarchyid, '/1/1/')
```

使用SET命令也可以做这个配置，或是从表中检索出一个hierarchyid值。

下一步，使用GetAncestor方法给第二个声明的hierarchyid变量赋值：

```
SELECT @New_WebpageLayoutID = @WebpageLayoutID.GetAncestor( 1 )
```

使用ToString方法来打印出变量被填充的值：

```
SELECT @New_WebpageLayoutID.ToString()
```

ToString方法将原生hierarchyid节点值转换为字符串表示。

如果想将字符串转换回实际的hierarchyid节点值，应该使用Parse方法。例如：

```
SELECT hierarchyid::Parse ('/1/1/')
```

这个查询返回：

```
0x5AC0
```

注意Parse绑定到hierarchyid的主类上，而不是绑定到实际的局部变量上。

14.3.3 返回子节点

GetDescendant方法返回指定层次位置的子节点。它有两个输入，且可以为NULL，或是指定某个节点的子节点来定义父级返回哪个子节点的边界。在这个例子中，查找路径/1/的后代节点：

```
DECLARE @WebpageLayoutID hierarchyid
DECLARE @New_WebpageLayoutID hierarchyid
SELECT @WebpageLayoutID = CONVERT(hierarchyid, '/1/')

SELECT @New_WebpageLayoutID = @WebpageLayoutID.GetDescendant(NULL,NULL)
SELECT @New_WebpageLayoutID.ToString()
```

这个查询返回：

```
/1/1/
```

解析

与上一个技巧相似，用variable.method格式对hierarchyid局部变量使用GetDescendant方法：

```
SELECT @New_WebpageLayoutID = @WebpageLayoutID.GetDescendant(NULL,NULL)
```

439

这个函数返回局部变量父级的子节点1/1/1。

14.3.4 返回节点的深度

GetLevel方法返回树中指定节点的深度。随后的示例返回/1/1/1/1/级别的整数值：

```
DECLARE @WebpageLayoutID hierarchyid
SELECT @WebpageLayoutID = CONVERT(hierarchyid, '/1/1/1/1/')

SELECT @WebpageLayoutID.GetLevel()
```

这个查询返回：

```
4
```

解析

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使用variable.method格式将GetLevel方法与hierarchyid局部变量连接起来：

```
SELECT @WebpageLayoutID.GetLevel()
```

这个函数返回局部变量节点深度的整数值。

14.3.5　返回根节点

GetRoot方法返回hierarchyid在树中的根值。这个函数不用与用户定义的hierarchyid局部变量进行连接。为了演示它，使用GetRoot方法从前面技巧中创建的表里获得根节点：

```
SELECT PageURL
FROM dbo.WebpageLayout
WHERE WebpageLayoutID = hierarchyid::GetRoot()
```

这个查询返回：

```
PageURL
http://joesack.com
```

解析

在这个技巧中，查询通过使用GetRoot方法返回根网页：

```
WHERE WebpageLayoutID = hierarchyid::GetRoot()
```

440

注意，GetRoot方法绑定到hierarchyid的主类上，而不是绑定到局部变量上。

14.3.6　确定节点是否为当前节点的子节点

IsDescendantOf方法返回真（1）和假（0）的布尔值指定节点是否为当前节点的子节点。这个方法使用一个输入参数来定义要检验的子节点，如下所示：

```
DECLARE @WebpageLayoutID hierarchyid
SELECT @WebpageLayoutID = CONVERT(hierarchyid, '/1/')

SELECT @WebpageLayoutID.IsDescendantOf('/')
SELECT @WebpageLayoutID.IsDescendantOf('/1/1/')
```

这个查询返回：

```
1

(1行受影响)

0

(1行受影响)
```

解析

这个技巧演示了使用IsDescendantOf方法来检验节点是否为当前节点的子节点。第一个IsDescendantOf方法检验了根节点是否为/1/的后代。由于是，所以返回1。第二个IsDescendantOf检验了1/1是否是/1/的后代。由于不是，所以返回0。

14.3.7　更改节点位置

GetReparentedValue方法返回给定两个输入参数新路径的节点值，两个输入参数分别为旧根和新根，如下所示：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DECLARE @WebpageLayoutID hierarchyid
DECLARE @New_WebpageLayoutID hierarchyid
SELECT @WebpageLayoutID = CONVERT(hierarchyid, '/1/1/')

SELECT @New_WebpageLayoutID = @WebpageLayoutID.GetReparentedValue('/1/', '/2/')
SELECT @New_WebpageLayoutID.ToString()
```

这个查询返回：

```
/2/1/
```

解析

这个技巧演示了使用GetReparentedValue方法更改既有节点的父节点。将@WebpageLayoutID局部变量中包含的hierarchyid值从/1/到/2转变为/2/1/。

441

14.4 原生空间数据

在SQL Server 2008之前，空间数据的数据存储需要使用包括创建自定义CLR程序集的非原生方法。SQL Server 2008引入了原生空间数据存储，提供了geography和geometry两种新数据类型。从根本上说，这些数据类型为位置和制图应用程序（比如微软Virtual Earth）和几何形状的表示提供了内建的功能。

Geometry数据类型表示平面球（欧几里得）坐标空间数据，也可以存储点、多边形、曲线和集合。

提示 geography或geometry数据类型上下文中的多边形可用来定义地球上区域和地区引用的位置。注意geography数据类型的数据在SQL Server 2008中仍然是二维的。

Geography数据类型用在圆球空间的存储上，可以存储坐标的经度、纬度、点、多边形、曲线和集合。这可能会经常用在制图或位置智能应用程序中。

SQL Server 2008支持熟知文本（WKT）、熟知二进制（WKB）和地理标记语言（GML）的XML传输格式来表示矢量几何映射对象。开放地理空间联盟（OGC）中常用这些格式，在本技巧中我将通过WKT格式联合使用geography数据类型。

提示 更多关于OGC和WKT/WKB/GML XML传送格式的标准文档请访问http://www.opengeospatial.org/standards/sfa 和 http://www.opengeospatial.org/standards/sfs。

下面两个技巧将演示如何存储和查询原生空间数据类型的数据。

14.4.1 存储空间数据

在这个技巧中，将演示如何以geography数据类型来存储空间数据。特别地，将创建存储明尼阿波利斯市的湖泊位置和形状的表。然后，将演示如何使用各种函数来处理geography数据。SQL Server 2008提供了执行几何和地理计算的几个内建函数。例如定义点、距离、地区之间的交集的计算能力。

先新建表，用来存储明尼阿波利斯市的湖泊位置和形状，且以geography数据类型来存储空间数据：

```
USE TestDB
GO

CREATE TABLE dbo.MinneapolisLake
   (MinneapolisLakeID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
   LakeNM varchar(50) NOT NULL,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    LakeLocationGEOG Geography NOT NULL,
    LakeLocationWKT AS LakeLocationGEOG.STAsText())
442 GO
```

注意，LakeLocationWKT列实际上是LakeLocationGEOG Geography列的计算列。geography数据类型的STAsText方法返回geometry列的WKT格式（我将在这个技巧的后面部分演示其不同）。

现在表已经创建好了，将插入表示3个湖泊的3行数据。然后，使用geography实例的Parse方法来输入每个湖泊多边形表示形式的WKT值。

警告　SQL Server 2008社区技术预览版（CTP）使用了geography数据类型用的WKT和WKB格式的纬度经度坐标顺序。到了RTM版，这个顺序被反转为使用经度纬度顺序。在INSERT或UPDATE操作中使用不正确的顺序将会引发错误。

Calhoun湖泊和Harriet湖泊都是圆形或椭圆形的形状，因此用4个经纬度对来定义多边形，即重复第一个经纬度对来完成多边形环：

```
-- Calhoun湖泊
INSERT dbo.MinneapolisLake
(LakeNM, LakeLocationGEOG)
VALUES ('Lake Calhoun',
geography::Parse('POLYGON((-93.31593 44.94821 , -93.31924 44.93603 ,
-93.30666 44.93577 , -93.30386 44.94321 , -93.31593 44.94821 ))'))

-- Harriet湖泊
INSERT dbo.MinneapolisLake
(LakeNM, LakeLocationGEOG)
VALUES ('Lake Harriet',
geography::Parse('POLYGON((-93.30776 44.92774 , -93.31379 44.91889 ,
-93.30122 44.91702 , -93.29739 44.92624 ,-93.30776 44.92774 ))'))
```

如果没有重复多边形环起点和终点，则会收到下面这个错误消息：

```
因为环的起点和终点不同，所以多边形输入无效。多边形每个环的起点和终点必须相同。
```

下一步，将插入表示Isles湖泊的多边形，与前面两个湖泊相比它的形状较不寻常，需要用7个不同的多边形经纬度对来表示（并且用第8个经纬度对来关闭多边形环）：

```
-- Isles湖泊（注意这里用了7个点，因为这个湖泊的形状很奇怪）
INSERT dbo.MinneapolisLake
(LakeNM, LakeLocationGEOG)
VALUES ('Lake of the Isles',
geography::Parse(
'POLYGON(( -93.30924 44.95847,
           -93.31291 44.95360,
           -93.30607 44.95178,
           -93.30158 44.95543,
           -93.30349 44.95689,
           -93.30372 44.96261,
           -93.3068 44.95720 8,
443        -93.30924 44.95847))'))
```

下面查询表中存储的数据：

```
SELECT LakeNM, LakeLocationGEOG, LakeLocationWKT
FROM dbo.MinneapolisLake
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回：

```
LakeNM          LakeLocationGEOG                                    LakeLocationWKT
Lake            Calhoun 0xE6100000010405000000BEDEFDF15E794640      POLYGON ((44.94821
                D1747632385457C09609BFD4CF774640CDE49B               -93.31593, 44.93603
                6D6E5457C0F949B54FC777464072BF4351A053               -93.31924, 44.93577
                57C04DA1F31ABB78464001A43671725357C0BE               -93.30666, 44.94321
                DEFDF15E794640D1747632385457C001000000               -93.30386, 44.94821
                02000000000100000FFFFFFFF0000000003                  -93.31593))
Lake Harriet    0xE6100000010405000000074982F2FC0764640             POLYGON ((44.92774
                16C1FF56B25357C08B4F01309E754640F5F3A6               -93.30776, 44.91889
                22155457C093E34EE960754640AC3940304753               -93.31379, 44.91702
                57C09F1F46088F764640130A1170085357C074               -93.30122, 44.92624
                982F2FC076464016C1FF56B25357C001000000               -93.29739, 44.92774
                02000000000100000FFFFFFFF0000000003                  -93.30776))
Lake of the Isles 0xE6100000010408000000 3B191C25AF7A4640           POLYGON ((44.95847
                5DA79196CA5357C099BB96900F7A4640D925AA               -93.30924, 44.9536
                B7065457C0527E52EDD37946405A12A0A69653               -93.31291, 44.95178
                57C003CFBD874B7A46402C4833164D5357C04A               -93.30607, 44.95543
                EF1B5F7B7A46406F2A52616C5357C0BA66F2CD               -93.30158, 44.95689
                367B464008C90226705357C098DD9387857A46               -93.30349, 44.96261
                40FAF202ECA35357C03B191C25AF7A46405DA7               -93.30372, 44.9572
                9196CA5357C001000000020000000001000000              -93.30688, 44.95847
                FFFFFFFF0000000003 -93.30924))
```

注意，LakeLocationGEOG中的原生geography数据并不可读——这就是为什么要使用STAsText方法定义的LakeLocationWKT计算列来显示可读的WKT格式文本。

解析

这个技巧演示了如何在geography数据类型中存储原生空间数据。首先，创建表来存储所有的湖泊。将LakeLocationGEOG列定义为geography数据类型：

```
LakeLocationGEOG Geography NOT NULL,
```

下面定义了引用geography列的计算列以及调用STAsText方法，用来显示geography实例可读的WKT格式：

```
LakeLocationWKT AS LakeLocationGEOG.STAsText())
```

接着，插入3个不同湖泊的行。在INSERT操作的VALUES参数中，使用了Parse命令（用geography::前缀，随后是多边形的WKT定义，以及相应定义湖泊边界的经纬度点）：

```
VALUES ('Lake Calhoun',
geography::Parse('POLYGON((-93.31593 44.94821, -93.31924 44.93603, -93.30666
44.93577, -93.30386 44.94321, -93.31593 44.94821))'))
```

POLYGON关键字表示了对象的类型（SQL Server 2008还能够以其他类型为实例，包括行字符串、多行字符串、多多边形、点和多点）。

444

注意，前面的代码片段中POLYGON实例类型的每个经纬度对都是用逗号分隔的。

提示 *geography数据类型也可以用来存储海拔和度量值，但是这些值不能用在任何计算上，只能以元数据进行保存。存储的数据被视为两维的。*

Parse方法与STGeomFromText方法等效，只是它提出一些关于空间引用ID（SRID）的假设。SRID

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

定义了用在geometry和geography数据类型的引用系统。在这里，geography数据类型的Parse将值为4326的SRID作为参数，它翻译为WGS84（世界测量系统1984）作为地理映射的引用。如果用STGeomFromText方法来替代Parse方法，要将第二个SRID参数指定为不同的空间引用类型。Sys.spatial_reference_论systems目录视图包含了所有通过SQL Server 2008认证的SRID。

14.4.2　查询空间数据

现在已经将geography值存储在表中，可以利用SQL Server 2008的原生计算功能来检索各种数据点。

提示　SQL Server 2008提供了一些处理geography和geometry数据类型的方法。本章演示了其中一部分，如果想查看完整的列表，请参考SQL Server联机丛书。

例如，可以使用STArea方法求出每个湖泊内部区域的面积，以及使用STLength方法求出总长：

```
SELECT LakeNM,
       LakeLocationGEOG.STLength() Length,
       LakeLocationGEOG.STArea() Area
FROM dbo.MinneapolisLake
```

这个查询返回：

```
LakeNM              Length              Area
Lake Calhoun        4330.60504437253    1131010.52091503
Lake Harriet        4010.50193580554    982158.281167269
Lake of the Isles   3473.37155877733    448936.179574728
```

现在，假设传统的双城马拉松赛计划从明尼阿波利斯（Minneapolis）市中心跑到表中某一个湖泊。如果有起点的经纬度（以点来表示），就可以使用STDistance方法求出明尼阿波利斯闹市区到每个湖泊的最短距离，如下所示（我会在“解析”部分对每一行进行讲解）：

```
DECLARE @DowntownStartingPoint geography
SET @DowntownStartingPoint = geography::Parse('POINT(-93.26319 44.97846)')

SELECT LakeNM,

       LakeLocationGEOG.STDistance(@DowntownStartingPoint) Distance
FROM dbo.MinneapolisLake
ORDER BY LakeLocationGEOG.STDistance(@DowntownStartingPoint)
```

445

这个查询返回明尼阿波利斯闹市区到每个湖泊的最短距离：

```
LakeNM              Distance
Lake of the Isles   3650.39170324889
Lake Calhoun        5063.86366294861
Lake Harriet        6400.09741657171
```

继续马拉松赛的示例，假定会有一架小飞机在从闹市区前往湖泊的运动员上方拉扯庆祝条幅。

目前为止，已经演示了点和多边形geography类型，不过在例子中的飞行路线需用Linestring来表示。在下一个查询中，将使用geography数据类型的STIntersects方法来判断飞机路线是否会和湖泊相交（飞过任何一个湖泊）：

```
DECLARE @FlightPath geography
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SET @FlightPath = geography::Parse('LINESTRING(-93.26319 44.97846, -93.30862
44.91695 )')

SELECT LakeNM,
       LakeLocationGEOG.STIntersects(@FlightPath) IntersectsWithFlightPath
FROM dbo.MinneapolisLake
```

结果显示从明尼阿波利斯闹市区的飞行路线只会和Harriet湖泊相交：

```
LakeNM                    IntersectsWithFlightPath
Lake Calhoun              0
Lake Harriet              1
Lake of the Isles         0
```

目前为止，没有在WHERE子句的上下文中使用任何geography数据类型方法。假设有比我在这里演示的更大的结果集，那么在搜索条件的上下文中使用这些空间方法就会带来非常明显的计算性能开销。为了解决这个问题，SQL Server 2008为geometry和geography数据类型提供了空间索引。

当查询在WHERE子句中使用空间索引时支持像STDistance和STIntersects的方法。空间索引通过过滤区域的底层网格表现形式来提升空间数据类型搜索性能。创建空间索引将空间平面分解为四级网格层次结构，这样就可以让SQL Server更快地过滤数据。

用CREATE SPATITAL INDEX命令来创建空间索引。CREATE SPATIAL INDEX的语法与常规的聚集或非聚集索引有相同的地方（一些共有的选项如PAD_INDEX、FILLFACTOR、DROP_EXISTING等）。如果计划在geometry数据类型上创建索引，在用BOUNDING_BOX选项定义索引时需要考虑不同的索引边界(geography边界是已经预先定义的)。还可以使用GRIDS选项来定义每个网格级别（四级网格）的密度，以及使用CELLS_PER_OBJECT选项指定每个定义的空间对象的单元格数。

继续这个技巧，现在将为前面创建的MinneapolisLake表中的geography列创建空间索引。

```
CREATE SPATIAL INDEX Spatial_Index_MinneapolisLake_LakeLocationGEOG
   ON dbo.MinneapolisLake(LakeLocationGEOG)
```

446

当前表中的行数很难证明索引的意义，不过，当表的规模增大时，使用索引会显著加速内部过滤过程，提升查询性能。如下所示。

提示 本技巧中最后一个查询因为表中只有很少的行，可能会对表产生聚集索引扫描。如果你再增加1 000行（表示湖泊）到表中，它将切换为在空间索引上进行聚集索引查找。

在本技巧的最后一个示例中，通过提供经纬度来确定我的船正在哪个湖泊上漂流：

```
DECLARE @LocationOfMyBoat geography
SET @LocationOfMyBoat= geography::Parse('POINT(-93.31329 44.94088)')

SELECT LakeNM
FROM dbo.MinneapolisLake
WHERE LakeLocationGEOG.STIntersects(@LocationOfMyBoat) = 1
```

这个查询返回：

```
LakeNM
Lake Calhoun
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

这个技巧演示了如何使用内建方法来对数据进行运算以及执行计算。第一个演示计算了每个多边形的长度（以米为单位）及相应的多边形面积。SELECT子句中仅需要引用geography数据类型列，随后是小数点，然后是方法名称（注意两个函数都没有输入参数，因此它们以空括号作为后缀）：

```
LakeLocationGEOG.STLength() Length,
LakeLocationGEOG.STArea() Area
```

接着，演示了如何计算从指定点到每个湖泊的距离，先声明要保存geography数据类型点的变量：

```
DECLARE @DowntownStartingPoint geography
```

下一步，使用geography::Parse方法为变量赋值，用一个经纬度对来定义点：

```
SET @DowntownStartingPoint = geography::Parse('POINT(-93.26319 44.97846)')
```

为变量赋好值，作为STDistance方法的输入参数来计算湖泊与输入参数的点之间的最短距离：

```
LakeLocationGEOG.STDistance(@DowntownStartingPoint) Distance
```

下一步，演示了如何计算飞过3个湖泊的路线的线表示方式的交叉点。首先，声明geography类型变量包含线字符串类型：

447

```
DECLARE @FlightPath geography
```

与前面的示例类似，在Parse方法中使用两个经纬度对来表示直线的两个端点来定义线字符串：

```
SET @FlightPath = geography::Parse('LINESTRING
(-93.26319 44.97846, -93.30862 44.91695 )')
```

在为变量赋好值之后，将它作为STIntersects方法的输入参数，这个方法与湖泊相交则返回1，不相交则返回0：

```
LakeLocationGEOG.STIntersects(@FlightPath) IntersectsWithFlightPath
```

下一步，讨论了在WHERE子句中使用空间方法时，使用空间索引来提升搜索性能。首先，使用CREATE SPATIAL INDEX命令，随后是用户定义的索引名称：

```
CREATE SPATIAL INDEX Spatial_Index_MinneapolisLake_LakeLocationGEOG
```

索引创建的第二行定义了表的名称，以及在圆括号内的空间数据类型列名：

```
ON dbo.MinneapolisLake(LakeLocationGEOG)
```

在这个语句中没有使用GRIDS选项。使用这个选项可以定义用来过滤数据的内部索引网格的每个级别的密度。选项有LOW（4×4网格）、MEDIUM（8×8网格）和HIGH（16×16网格）。默认值为MEDIUM，因此默认情况下索引在内部索引网格结构的四个级别中都使用8×8网格。

一旦索引被映射到四级网格层次结构中，SQL Server读取空间列数据和执行的过程被称为分割过程。分割过程定义了空间对象接触了多少网格单元——它允许索引在定义的空间中更高效地定位对象。CREATE SPATIAL INDEX有CELLS_PER_OBJECT选项，它定义了每个空间对象内部单元的数量范围（即从1到8192）。默认为16，这也是索引使用的数量。每个对象有越多的单元，空间索引就需要越多的存储空间。

在创建索引之后，演示了在SELECT查询的WHERE子句中使用STIntersects方法。首先，定义空间点类型，表示在湖泊中小船的位置：

```
DECLARE @LocationOfMyBoat geography
SET @LocationOfMyBoat= geography::Parse('POINT(-93.31329 44.94088)')
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

为变量完成赋值之后，将它作为STIntersects方法的输入——查询小船与湖泊在哪里相交的所有行：

```
WHERE LakeLocationGEOG.STIntersects(@LocationOfMyBoat) = 1
```

这个查询返回小船与其相交的湖泊的名称，即Calhoun湖泊，由于表里插入了10 000个湖泊（即10 000行数据），空间索引将会显著提升查询性能。 448

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第15章 提 示

SQL Server的查询优化过程负责在SELECT查询执行的时候产生查询执行计划。SQL Server会有代表性地选择一个高效的计划来取代低效的那个。如果情况不是这样的，就需要检验查询执行计划、表统计信息、支撑的索引以及其他因素，这些在第28章都会有更详细的介绍。

最后在研究了查询性能之后，可能会决定使用提示（hint）来覆盖SQL Server查询优化器决定的过程。

警告 应该在绝大多数情况下让SQL Server查询优化器不在提示的辅助下来决定查询执行计划。就算提示短期内有效，也要记住在将来由于数据库更改了内容，使用的查询计划应该会更高效，但是因为指定的提示覆盖了优化器，这种情况并不会出现。而且，当SQL Server发布了新的补丁或版本后，有效的提示也可能会改变。

我将复习SQL Server中可用的不同类型的提示，包括新的SQL Server 2008的提示FORCESEEK，可以使用它将索引查找来替换索引扫描。在大多数情况下，新的提示不是必需的，然而，在某些条件下可能会遇到这种情况：当期望在索引中使用索引查找时查询优化器并没有选择它。

15.1 使用联结提示

联结“提示”在这里可能有点儿用词不当，因为联结提示会强制查询优化器来使用你命令的方式来联结表。联结提示强制使用内部JOIN操作在查询中联结两个表。表15-1描述了可用的联结提示。

表15-1 联结提示

提示名	描 述
LOOP	当一个表很小另外一个表很大并且在联结的列上有索引的情况下，LOOP联结工作得最好
HASH	HASH联结对大的未排序的表最理想
MERGE	MERGE联结对根据联结列排序的中等的或者大的表最理想
REMOTE	REMOTE强制联结操作在右边引用的表（JOIN子句中第二个引用的表）那里产生。为了有好的性能，左表应该是本地表并且应该比远程的右表有更少的行

强制 HASH 联结

在演示联结提示如何工作之前，示例先进行一个原始的、非提示查询：

```
-- (有关SHOWPLAN_TEXT的更多内容，请参阅第28章)
SET SHOWPLAN_XML ON
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
GO

SELECT p.Name,
      r.ReviewerName,
      r.Rating
FROM   Production.Product p
INNER  JOIN Production.ProductReview r ON
   r.ProductID = p.ProductID
GO

SET SHOWPLAN_XML OFF
GO
```

这个查询返回如下部分结果（SHOWPLAN_TEXT返回查询将会如何进行的相关信息，但不是真正执行查询）：

```
<RelOp NodeId="0" PhysicalOp="Nested Loops" LogicalOp="Inner Join"
EstimateRows="4" EstimateIO="0" EstimateCPU="1.672e-005" AvgRowSize="82"
EstimatedTotalSubtreeCost="0.0115181" Parallel="0" EstimateRebinds="0"
EstimateRewinds="0">
```

下面的示例提交相同的查询，但这次使用了联结提示：

```
SET SHOWPLAN_XML ON
GO

SELECT p.Name,
      r.ReviewerName,
      r.Rating
FROM   Production.Product p
INNER HASH JOIN Production.ProductReview r ON
   r.ProductID = p.ProductID
GO

SET SHOWPLAN_XML OFF
GO
```

450

这个查询返回如下的部分结果：

```
<RelOp NodeId="0" PhysicalOp="Hash Match" LogicalOp="Inner Join" EstimateRows="4"
EstimateIO="0" EstimateCPU="0.0350031" AvgRowSize="82"
EstimatedTotalSubtreeCost="0.0443511" Parallel="0" EstimateRebinds="0"
EstimateRewinds="0">
```

解析

第一次，在查询中没有提示，SET SHOWPLAN_TEXT用于查看查询如何被SQL Server执行。

SET SHOWPLAN_XML返回查询执行计划的XML译文。逻辑联结（INNER JOIN）操作的RelOp片断的摘要显示了嵌套循环物理联结操作。

在第二个查询中，使用提示来强制嵌套循环联结来执行散列联结操作。通过在INNER和JOIN关键字中加HASH来实现：

```
INNER HASH JOIN Production.ProductReview r ON
   r.ProductID = p.ProductID
```

现在在第二个SHOWPLAN_TEXT的结果中可以看到，查询使用了散列联结来联结两个表。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

15.2　使用查询提示

一些查询提示和前面技巧中讨论的联结提示差不多，都是查询的一种指示，它们会覆盖SQL Server查询优化器作出的决定。使用查询提示可能会在短期内满足你当前的情况，但随着时间推移不会总是最有效率的。尽管如此，只要进一步理解查询优化器自动作出的选择，你以后就可以自己决定如何来使用它们。

查询提示能用于SELECT、INSERT、UPDATE和DELETE语句中，如表15-2的描述。

表15-2　查询提示

提　示　名	描　　述
{HASH \| ORDER} GROUP	和GROUP BY子句一起使用，为GROUP BY和COMPUTE聚合指定使用散列或者排序
{CONCAT \| HASH \| MERGE} UNION	为UNION操作指定用于联结结果集的策略
{LOOP \| MERGE \| HASH} JOIN	强制所有联结操作在整个查询中执行循环、合并或者散列联结
FAST integer	提高开始整数值指定的那些行的获取速度
FORCE ORDER	如果指定了，那么表联结的时候按照表出现的次序执行
MAXDOP number_of_processors	覆盖查询的max degree of parallelism服务器配置选项
OPTIMIZE FOR (@variable_name = literal_constant) [,...n]	在编译和优化查询时提示SQL Server对变量使用一个特定变量值或多个值。例如，你可以插入一个字面量常量，它跨期望参数的范围返回最佳性能
ROBUST PLAN	以表的行大小达到最大为假设创建查询计划
KEEP PLAN	“放宽”对查询重新编译的阈值
KEEPFIXED PLAN	强制查询优化器不因为统计数据或索引列的修改而重新编译查询。只有架构的变动或者sp_recompile才会引起查询计划被重新编译
EXPAND VIEWS	防止查询优化器对引用的表使用索引视图
MAXRECURSION number	指定查询允许的最大递归（1到32 757）。如果选择0，则没有限制。默认递归限制是100，这个选项和公共表表达式（CTE）一起使用
USE PLAN 'xml_plan'	USE PLAN让SQL Server使用你认为能让查询执行得更好的查询计划（xml_plan中提供的字面量值）。更多信息请参阅第28章
PARAMETERIZATION { SIMPLE \| FORCED }	与新的PARAMETERIZATION数据库设置有关，它控制查询是否参数化（查询中的字面量值被缓存查询计划中的参数替代）。如果选择了PARAMETERIZATION SIMPLE，SQL Server会决定哪个查询参数化，哪个不参数化。如果使用PARAMETERIZATION FORCED，这个数据库中的所有查询都会参数化。有关数据库设置的更多信息，请参阅第28章
RECOMPILE	强制SQL Server在查询执行后丢弃执行计划。也就是说下次查询执行的时候，会被强制重新编译查询计划。尽管SQL Server通常会重用有效的查询计划——有些时候低效的查询计划会被重用。重新编译强制SQL Server发出一个新的计划（但重新编译也会有更多开销）

强制语句重编译

这个示例使用RECOMPILE查询提示来重新编译查询，在执行后强制SQL Server放弃为查询生成的计划。使用RECOMPILE查询提示后，下次相同或相似的查询执行后会生成一个新的计划。你可能会希望对一些易变的查询计划使用RECOMPILE，对于那些查询，相同计划的不同搜索条件值会导致返回的行数变

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

化很大。对于这种情况，使用编译的查询计划会对查询性能不利。缓存的以及重用的查询执行计划（避免编译的开销）可能偶尔会超过新的、重新编译的计划。 452

注解 SQL Server使用语句级别的重编译。和整个批处理重新编译不一样，当相关表添加索引或者修改数据后，仅批处理中的改变所影响的各个语句才会被重新编译。

一般情况下，你会希望在存储过程中使用RECOMPILE查询提示。这样就可以控制哪个语句自动重新编译，而不是必须重新编译整个存储过程。

这个技巧先执行如下没有查询提示的查询：

```
DECLARE @CarrierTrackingNumber nvarchar(25) = '5CE9-4D75-8F'

SELECT   SalesOrderID,
   ProductID,
   UnitPrice,
   OrderQty
FROM Sales.SalesOrderDetail
WHERE CarrierTrackingNumber = @CarrierTrackingNumber
ORDER BY   SalesOrderID,
        ProductID
```

这个查询返回：

```
SalesOrderID    ProductID    UnitPrice        OrderQty
47964           760          469.794          1
47964           789          1466.01          1
47964           819          149.031          4
47964           843          15.00            1
47964           844          11.994           6

(5行受影响)
```

现在可以查询动态管理视图sys.dm_exec_cached_plans（关于DMV请看第22章），来查看这个查询产生的内存中编译的计划是否可以被重用：

```
SELECT cacheobjtype, objtype, usecounts
FROM sys.dm_exec_cached_plans
CROSS APPLY sys.dm_exec_sql_text(plan_handle)
WHERE text LIKE 'DECLARE @CarrierTrackingNumber%'
```

这个查询返回缓存对象类型和查询执行使用这个对象的次数：

```
cacheobjtype        objtype        usecounts
Compiled Plan       Adhoc          1

(1行受影响)
```

453

下一步，再次执行原来的查询来测试缓存的计划是否可以重用。执行原始查询后查询sys.dm_exec_cached_plans，将看到如下的增长数量，意思是重用了编译的Adhoc计划：

```
cacheobjtype        objtype        usecounts
Compiled Plan       Adhoc          2
```

现在演示RECOMPLILE提示。在做这个之前，先清空过程缓存（不要在运营环境的SQL Server实例

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

尝试它——只能在测试环境下使用它）：

```
DBCC FREEPROCCACHE
```

它清空了缓存，现在指示SQL Server不要创建或重用既有计划：

```
DECLARE @CarrierTrackingNumber nvarchar(25) = '5CE9-4D75-8F'

SELECT   SalesOrderID,
   ProductID,
   UnitPrice,
   OrderQty
FROM Sales.SalesOrderDetail
WHERE CarrierTrackingNumber = @CarrierTrackingNumber
ORDER BY   SalesOrderID,
         ProductID
OPTION (RECOMPILE)
```

再次查询sys.dm_exec_cached_plans，没有看到计划被重用：

```
cacheobjtype objtype usecounts

(0 行受影响)
```

解析

这个查询演示了查询提示的使用，在查询最后的OPTION子句中引用查询提示：

```
OPTION (RECOMPILE)
```

没有提示的情况下，原始查询会产生可在后续执行中重用的缓存查询计划。为它加上提示后，通过在每次执行查询时强制查询重编译新的计划来覆盖这个行为。

值得重复的是SQL Server对于处理查询时作出的决定在大多数时候值得信赖。查询提示可以提供对例外情况的更多控制权，我们可以通过它来覆盖SQL Server的选择。

15.3　使用表提示

表提示和查询提示相似，能用于覆盖SELECT、INSERT、UPDATE和DELETE的默认处理行为。可以为一个查询使用多个表提示，并使用逗号分隔，只要它们不属于同一个分组就可以。务必测试带查询提
454 示和不带查询提示情况下的查询性能（更多有关查询性能的信息请参阅第28章）。

表15-3列出了可用的表提示。一些提示不能一起使用，所以在表中进行了分组。例如，不能对同一个查询同时使用NOLOCK和HOLDLOCK。

表15-3　表提示

提　示　名	描　　述
FASTFIRSTROW	这个提示不再建议使用，并会在将来SQL Server的版本中移除
FORCESEEK	SQL Server 2008新提供的，这个提示允许强制查询只考虑对指定表或视图的聚集索引或非聚集索引查找访问路径，而不使用扫描。你可以把它交给优化器来让其选择做索引查找，或者你也可以和INDEX提示一起使用FORCESEEK（下面会描述）
INDEX (index_val [,... n])	这个提示覆盖SQL Server的索引选择并且强制表使用某个指定的索引
NOEXPAND	如果引用了索引视图，查询优化器将不会访问用来定义视图的数据对象的数据结构。只有索引视图上的索引会用在查询上

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

提示名	描述
HOLDLOCK，SERIALIZABLE，REPEATABLEREAD，READCOMMITTED，READCOMMITTEDLOCK，READUNCOMMITTED，NOLOCK	选择这些提示中的一个来确定表使用的隔离级别。指定NOLOCK意味着操作（例如SELECT）对表不加锁
ROWLOCK，PAGLOCK，TABLOCK，TABLOCKX	这个提示指定表加锁的粒度，例如，选择ROWLOCK来强制对查询操作只使用行锁
READPAST	这个提示跳过加锁的行且不读取它们
UPDLOCK	这个提示会强制产生更新锁而不是共享锁（和NOLOCK或XLOCK不兼容）
XLOCK	这个提示强制对引用的资源使用排他锁（和NOLOCK或UPDLOCK不兼容）
KEEPIDENTITY	这个选项应用到OPENROWSET函数的BULK插入功能（参阅第27章），并且影响行如何插入到一个有IDENTITY列的表中。如果你使用这个提示，SQL Server会使用数据文件的标识值，而不是生成自己的。更多有关IDENTITY列的信息，请参阅第4章
KEEPDEFAULTS	和KEEPIDENTITY差不多，这个表提示也应用到OPENROWSET函数。使用这个表提示指定不包含在批量加载操作中的列会赋值为列默认值。更多有关默认列的信息，请参阅第4章
IGNORE_CONSTRAINTS	另外一个OPENROWSET提示，在导入数据的时候IGNORE_CONSTRAINTS让SQL Server忽略CHECK约束。更多有关CHECK约束的信息请参阅第4章
IGNORE_TRIGGERS	在使用OPENROWSET的BULK选项时，这个查询提示不让INSERT触发器触发

455

15.3.1 不锁定执行查询

这个示例从Production.Document表返回所有Status列等于1的行的DocumentID和Title。它使用NOLOCK表提示，意味着查询不会在Production.Document表上放置共享锁：

```
SELECT DocumentID,
       Title
FROM Production.Document
WITH (NOLOCK)
WHERE Status = 1
```

解析

本例的关键是WITH关键字，它在圆括号内使用NOLOCK表提示：

```
WITH (NOLOCK)
```

NOLOCK让查询不在被影响的行或数据页上放置共享锁——允许你在不被阻塞或不阻塞其他查询的情况下读取（但是会遇到“脏读”问题）。

15.3.2 强制 SEEK 覆盖 SCAN

SQL Server 2008引入新的FORCESEEK表提示，可以用来将索引扫描替换为索引查找。在大多数情况下，这个提示不是必要的。不过，在某些条件下，你可能会遇到当期望在索引中使用索引查找时查询优化器并没有选择它的情况。

会有一些原因导致产生不良的查询计划。例如，如果表数据经常变化并且统计信息已不再准确，就会产生不良的计划。另一个示例是带有拙劣结构WHERE子句的查询没有为查询优化器过程提供足够的或有用的信息。

15

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果只是为了执行对指定数据的单独查找，而你发现在检索一行之前查询扫描了整个索引，这样的扫描带来的I/O开销是显著的（尤其是非常大的表）。在尽职工作的情况下（更新统计信息，优化查询），你可能会考虑使用新的FORCESEEK表提示。

随后的查询使用SET SHOWPLAN XML来返回估计的查询执行计划。有关SET SHOWPLAN XML更详细的信息请参阅第28章。在这个技巧中，首先打开SHOWPLAN XML，然后对Production.TransactionHistory表执行聚集索引扫描的查询：

```
SET SHOWPLAN_XML ON
GO

SELECT DISTINCT TransactionID, TransactionDate
FROM Production.TransactionHistory
WHERE ReferenceOrderID BETWEEN 1000 AND 100000
GO

SET SHOWPLAN_XML OFF
```

这个查询返回估计的查询执行计划。可以从随后的嵌入在XML结果的片断中看到使用了聚集索引扫描操作：

456

```
<RelOp NodeId="0" PhysicalOp="Clustered Index Scan" LogicalOp="Clustered Index Scan"
EstimateRows="112121" EstimateIO="0.586088" EstimateCPU="0.124945" AvgRowSize="23"
EstimatedTotalSubtreeCost="0.711033" Parallel="0" EstimateRebinds="0"
EstimateRewinds="0">
```

引用的表Production.TransactionHistory在TransactionID列上有一个聚集索引，在ProductID列上有一个非聚集索引，以及在ReferenceOrderID和ReferenceOrderLineID列上有一个复合索引。在这个情况下，ReferenceOrderID值的范围可能是扫描的原因，假设这个示例是想使用查找操作。例如，如果实际表中值的范围从1000到100000只会返回几行，但由于某些原因索引的统计信息表明的却是不同的内容，就可以使用FORCESEEK来完成一个不同的数据访问方式。

警告 这个示例只是为了说明问题。下一个查询的强制查找也不是最优的。

```
SET SHOWPLAN_XML ON
GO

SELECT DISTINCT TransactionID, TransactionDate
FROM Production.TransactionHistory WITH (FORCESEEK)
WHERE ReferenceOrderID BETWEEN 1000 AND 100000
GO

SET SHOWPLAN_XML OFF
```

这次你会从估计的查询计划XML输出中看到使用了索引查找操作：

```
<RelOp NodeId="3" PhysicalOp="Index Seek" LogicalOp="Index Seek"
EstimateRows="112121" EstimateIO="0.15794" EstimateCPU="0.0617452" AvgRowSize="11"
EstimatedTotalSubtreeCost="0.219685" Parallel="1" EstimateRebinds="0"
EstimateRewinds="0">
```

你也会在XML输出中看到使用了指定的索引是由ReferenceOrderID和ReferenceOrderLineID列构成的非聚集索引：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<Object Database="[AdventureWorks]" Schema="[Production]"
Table="[TransactionHistory]"
Index="[IX_TransactionHistory_ReferenceOrderID_ReferenceOrderLineID]" />
```

在前面的查询中，允许SQL Server选择通过哪个索引做查找。不过，也可以选择强制查找操作并指定使用哪个索引。例如：

```
SELECT DISTINCT TransactionID, TransactionDate
FROM Production.TransactionHistory WITH (FORCESEEK, INDEX
(IX_TransactionHistory_ReferenceOrderID_ReferenceOrderLineID))
WHERE ReferenceOrderID BETWEEN 1000 AND 100000
```

457

解析

FORCESEEK命令是一个应该很少使用的查询提示，但在可以确定SQL Server选择了非最优访问路径的情况下非常有用。FORCESEEK可以用在SELECT、UPDATE或DELETE的FROM子句中。

在这个技巧中，通过WITH关键字和随后的在圆括号中的提示名称引用了表提示：

```
FROM Production.TransactionHistory WITH (FORCESEEK)
```

它覆盖了查询的原始聚集索引扫描访问路径。如果在表上有多个索引，单独使用这个提示会告知SQL Server来从索引中选出最佳查找。不过，也可以通过指定INDEX提示缩小使用范围，如下所示：

```
FROM Production.TransactionHistory WITH (FORCESEEK, INDEX
(IX_TransactionHistory_ReferenceOrderID_ReferenceOrderLineID))
```

INDEX提示随后是圆括号内的索引名称，也可以指定索引号。对于FORCESEEK，如果不是必须这样做，通常不需要指定INDEX提示，因为在那一刻SQL Server会选出最优的可用查找路径。

458

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第16章 错误处理 16

在本章中，将介绍使用T-SQL创建、引发和处理SQL Server错误的技巧。

16.1 系统定义和用户定义的错误消息

第一组是关于查看和引发系统和用户定义的错误消息的技巧。sys.messages表包含SQL Server实例中所有用户定义和内建的错误消息的行。内建错误消息（build-in error message）是那些为响应标准SQL Server错误而引发的消息。用户定义错误消息（user-defined error message）经常在第三方应用程序中使用，应用程序为了在其内部使用而定义一组错误消息。用户定义错误消息允许参数化（parameterization），意思是你可以创建允许基于参数对消息进行自定义的自定义消息（正如在本章后面RAISERROR的讨论中说明的那样）。

16.1.1 查看系统错误信息

使用系统目录视图sys.messages可以查看SQL Server实例中所有的系统和用户定义的错误消息，正如这个示例所演示的：

```
SELECT message_id, severity, is_event_logged, text
FROM sys.messages
ORDER BY severity DESC, text
```

它返回如下部分结果（为了使显示清晰，对输出进行了截断和格式化）：

message_id	severity	is_event_logged	text
832	24	1	A page that should have been constant has changed (expected checksum: %08x, actual checksum: %08x, database %d, file '%ls', page %S_PGID). This usually indicates a memory failure or other hardware or OS corruption.
1459	24	1	An error occurred while accessing the database mirroring metadata. Drop mirroring (ALTER DATABASE database_name SET PARTNER OFF) and reconfigure it.
17405	24	1	An image corruption/hotpatch detected while reporting exceptional situation. This may be a sign of a hardware problem. Check SQLDUMPER_ERRORLOG.log for details.

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个技巧中，简单的SELECT查询返回了如下关于SQL Server内建错误消息和为这个特定SQL Server实例定义的自定义错误消息的信息。

- message_id。这是错误消息标识符。
- severity。这是严重级别。
- is_event_logged。用于标记错误是否写入Windows事件日志。
- text。这是消息的文本。

严重级别范围为1～25，使用如下隐含的分类。

- 严重级别0～10表示信息性消息。
- 严重级别11～16是用户可以纠正的数据库引擎错误（当执行查询时丢失的数据库对象、不兼容的锁定提示、事务死锁、权限拒绝以及语法错误）。例如，违反PRIMARY KEY将返回严重级别为14的错误。被零除错误返回严重级别为16的严重级别错误。
- 严重级别17～19是需要系统管理员注意的错误（例如，SQL Server返回内存资源不足，或数据库引擎已达到极限）。
- 严重级别20～25是致命错误和系统问题（影响数据库的硬件或软件损坏、完整性问题和媒体故障）。

sys.messages的文本字段包含从数据库引擎中呈现给用户的实际错误消息。注意，技巧的结果中的第一个消息有百分号和其他符号的组合：

```
A page that should have been constant has changed (expected checksum: %08x, actual
checksum: %08x, database %d, file '%ls', page %S_PGID). This usually indicates a
memory failure or other hardware or OS corruption.
```

%是允许数据库引擎基于当前数据库上下文和错误事件来自定义错误消息输出的替换参数。拼接到%的值表示数据类型和替换参数的长度。

16.1.2 创建用户定义错误消息

在这个技巧中，演示了如何使用系统存储过程sp_addmessage创建新的用户定义错误消息。你可能会希望为你使用的应用程序创建用户定义的自定义消息，以确保特定于应用程序的错误处理程序的一致性。创建新错误消息，将它添加到系统目录视图sys.messages，并且允许使用RAISERROR命令（将在下一技巧中回顾）调用它。

这个系统存储过程的语法如下：

```
sp_addmessage [ @msgnum = ] msg_id ,
    [ @severity = ] severity ,
    [ @msgtext = ] 'msg'
    [ , [ @lang = ] 'language' ]
    [ , [ @with_log = ] 'with_log' ]
    [ , [ @replace = ] 'replace' ]
```

表16-1简要地描述了参数。

460

表16-1 sp_addmessage的参数

参　数	描　述
msg_id	这是用户提供的错误ID，范围是50 001到2 147 483 647。这个消息ID不是表的唯一键或主键，唯一组合键是消息ID和语言ID的组合

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

参　数	描　述
severity	这个参数定义消息的严重级别（1～25）
msg	这个参数表示实际的错误消息，它使用的数据类型是nvarchar(255)
language	这个参数指定错误消息使用的语言
with_log	这个参数定义当错误发生时，消息是否写入Windows应用程序日志
replace	当指定它时，既有的用户定义错误（基于消息ID和语言ID）会被传入系统存储过程的新参数覆盖

在这个技巧中，将创建新的错误消息来警告用户，他们的组不能更新特定的表（例如，你正在数据库中建立基于应用程序的安全系统时可能使用的表）：

```
-- 创建新消息
USE master
GO

EXEC sp_addmessage
      100001,
      14,
   N'The current table %s is not updateable by your group!'
GO

-- 使用新消息（RAISERROR将在下一技巧中介绍）
RAISERROR (100001, 14, 1, N'HumanResources.Employee')
```

这个查询返回：

```
消息100001，级别14，状态1，第1行
The current table HumanResources.Employee is not updateable by your group!
```

解析

在这个技巧中，使用sp_addmessage创建新的消息：

```
EXEC sp_addmessage
      100001,
      14,
   N'The current table %s is not updateable by your group!'
```

第一个参数100001是新消息的ID，其可以使用50 001与2 147 483 647之间的整数值。第二个参数14指定了严重级别，第三个参数是实际的错误消息。

在消息的主体中包括了替换参数值%s，s说明参数是字符串值。也可以指定参数为有符号整数（d或i）、无符号八进制数（o）、无符号整数（u）或无符号十六进制数（x或X）。

461

没有使用其他可选的参数，比如language、with_log和replace。这个技巧中最后的命令RAISERROR用来引发新创建错误的实例：

```
RAISERROR (100001, 14, 1, N'HumanResources.Employee')
```

RAISERROR经常用来返回与应用程序或业务逻辑有关的错误，例如返回这样的错误，它们虽然在语法上是正确的，但是违反了应用程序或业务的某个条件或需求。

在这个示例中，第一个参数是新的错误消息ID，第二个参数是严重级别，第三个参数是状态（可以用来识别代码的哪一部分出现了错误的编号），第四个是传入到错误消息的Unicode替换参数。参数可以为int、tinyint、smallint、varchar、char、nchar、nvarchar、varbinary和binary数据类型

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的替换参数。然后，将HumanResurces.Employee值插入到替换参数值后的新错误消息返回给SQL用户。

16.1.3 删除用户定义错误消息

在这个技巧中，我演示了如何从sys.messages表中删除用户定义错误消息。语法如下：

```
sp_dropmessage [ @msgnum = ] message_number
[ , [ @lang = ] 'language' ]
```

表16-2简要地描述了参数。

表16-2 sp_dropmessage的参数

参 数	描 述
message_number	这是用户定义错误消息的消息号
language	这是要删除的消息的语言。如果你指定ALL，并且存在具有相同消息号但是不同语言的消息，则这个编号的所有消息都将被删除

这个技巧删除前面技巧中创建的用户定义错误消息：

```
EXEC sp_dropmessage 100001
```

解析

这个技巧使用系统存储过程sp_dropmessage删除了前面技巧创建的用户定义错误消息。这个系统存储过程只可以用来删除用户添加的消息，即消息ID大于49 999的消息。

16.2 手动引发错误

RAISERROR命令允许调用系统目录视图sys.messages中的用户定义错误消息，或者从字符串或字符串变量产生的错误消息。语法如下：

```
RAISERROR ( { msg_id | msg_str | @local_variable }
    { ,severity ,state }
    [ ,argument [ ,...n ] ] )
    [ WITH option [ ,...n ] ]
```

462

表16-3简要地描述了该命令的参数。

表16-3 RAISERROR的参数

参 数	描 述
msg_id \| msg_str \| @local_variable	当使用RAISERROR时，你可以选择这个参数的3个选项中的一个。msg_id选项是sys.messages表中的用户定义错误消息号。msg_str是最多有2 047个字符的用户定义消息。@local_variable是用来传递这个消息字符串的字符串变量
severity	这个参数定义消息的严重级别（1～25）
state	这个参数指定1～127的用户定义编号，可以用来标识失败代码的位置（例如，假设你的代码被分为多个部分）
argument [,...n]	这个是用在错误消息中的一个或多个替换参数
WITH optin [,...n]	在WITH子句中允许3个选项，包括LOG、NOWAIT和SETERROR。LOG将写入SQL Server错误和Windows应用程序日志。NOWAIT立即将消息发送到客户端。SETERROR将@@ERROR和ERROR_NUMBER值设置到错误消息ID中（如果没有使用sys.messages中的错误，则为50 000）

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

调用错误消息

在这个技巧中，创建一个存储过程来向表HumanResources.Department INSERT一个新行。当尝试将新部门插入到表HumanResources.Department中时，首先验证组名称，查看是否为Research and Development。如果不是，将不会发生插入操作，并且将使用RAISERROR引发一个错误。

```
USE AdventureWorks
GO

CREATE PROCEDURE dbo.usp_INS_Department
   @DepartmentName nvarchar(50),
    @GroupName nvarchar(50)
AS
IF  @GroupName = 'Research and Development'
BEGIN
   INSERT HumanResources.Department
    (Name, GroupName)
   VALUES (@DepartmentName, @GroupName)
END
ELSE
BEGIN
RAISERROR('%s group is being audited
for the next %i days.
No new departments for this group can be added
during this time.',
           16,
           1,
           @GroupName,
           23)
END

GO
```

463

下一步，执行这个新过程：

```
EXEC dbo.usp_INS_Department 'Mainframe Accountant', 'Accounting'
```

这个查询返回：

```
消息50000，级别16，状态1，过程usp_INS_Department，第13行
Accounting group is being audited
for the next 23 days.
No new departments for this group can be added during this time.
```

在存储过程中创建错误消息的另一个可选的方法是将它创建为用户定义消息（如本章前面所讨论的）。例如：

```
EXEC sp_addmessage
      100002,
      14,
   N'%s group is being audited for the next %i
days. No new departments for this group can be added
during this time.'
GO
```

然后，通过重写前面的RAISERROR示例，你可以引用用户定义错误消息号：

```
...
ELSE
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
BEGIN

   RAISERROR(100002,
             16,
             1,
             @GroupName,
             23)
END
```

解析

如果没有满足特定的IF条件，这个技巧将使用RAISERROR返回错误。RAISERROR经常用来从T-SQL批处理、存储过程和触发器中发送错误到调用的应用程序，特别是对于通常不会引发语法错误的数据或逻辑条件。

在存储过程的主体中，验证了组名称的值，如果它等于Research and Development，则将发生插入操作：

```
IF   @GroupName = 'Research and Development'
BEGIN
   INSERT HumanResources.Department
   (Name, GroupName)
   VALUES (@DepartmentName, @GroupName)
END
```

因为组不等于Research and Development，所以ELSE子句启动了RAISERROR命令：

```
ELSE
BEGIN
   RAISERROR('%s group is being audited for the next %i
days. No new departments for this group can be added
during this time.',
             16,
             1,
             @GroupName,
             23)
END
```

RAISERROR命令的第一个参数是错误消息文本，它使用了两个替换参数：一个是组名称，另一个是要审计组的天数。第二个参数16是严重级别。第三个参数1是状态。最后两个参数@GroupName和23是引发错误时插入到错误消息的替换参数。 464

这个技巧也演示了添加用户定义错误消息，然后用RAISERROR引发它，而不是直接创建文件。这个技术对于必须在数据库中的多个地方使用的错误消息非常有用，并且借助它可以不用在每个引用它的过程或脚本中重新输入消息。它同时也确保了错误消息的一致性。

16.3 捕获和处理应用程序错误

可以使用TRY...CATCH命令在T-SQL代码中捕捉执行错误。TRY...CATCH可以捕捉任何严重级别大于10的执行错误（只要引发的错误没有强制终止T-SQL用户会话）。TRY...CATCH也可以处理使用RAISERROR调用的（大于10的）严重级别错误。

TRY...CATCH的语法如下：

```
BEGIN TRY
    { sql_statement | statement_block }
END TRY
BEGIN CATCH
    { sql_statement | statement_block }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
END CATCH
```

用在TRY和CATCH部分的参数是sql_statement和statement_block。简而言之，TRY块中的语句是你希望执行的语句。如果在TRY块中引发了错误，就会执行CATCH块的代码。CATCH块用来处理错误。处理的意思是你希望采取一些操作以响应错误：要么是报告错误的信息，要么是将信息记录在错误表中，或是回滚打开的事务。

使用TRY...CATCH的好处是可以在代码块中嵌套错误处理，这样可以更优雅地、并且以比非TRY...CATCH方式更少的代码处理错误。TRY...CATCH也允许使用SQL Server错误记录和事务状态函数，
465 它可以捕捉错误事件的粒度错误信息。表16-4详细地描述了这些参数的使用。

表16-4　错误和事件状态函数

参　数	描　述
ERROR_LINE	这个参数定义了在SQL语句或块中引发错误的错误行号
ERROR_MESSAGE	这个参数是在SQL语句或块中引发的错误消息
ERROR_NUMBER	这个参数是在SQL语句或块中引发的错误编号
ERROR_PROCEDURE	这个参数定义了引发错误的触发器或存储过程的名称（假定TRY...CATCH用在存储过程或触发器中）
ERROR_SEVERITY	这个参数表示在SQL语句或块中引发错误的严重级别
ERROR_STATE	这个参数指定了在SQL语句或块中引发错误的状态
XACT_STATE	在CATCH块中，XACT_STATE报告在TRY块中打开的事务的状态。如果返回0，则说明TRY块中没有打开的事务。如果返回1，则意味着在TRY块中没有引发错误。如果返回-1，说明TRY块中发生了错误，并且事务必须回滚。XACT_STATE也可以用在TRY...CATCH命令之外的地方

如果在TRY批处理中遇到了错误，SQL Server将在错误发生的位置退出，并且移到CATCH块，而不继续执行TRY批处理中的任何其他语句（如果你使用嵌套的是TRY...CATCH块，就不遵循这个规则，本章稍后演示这一点）。

TRY...CATCH可以用在触发器或存储过程中，也可以用来封装一个存储过程的实际执行（捕捉过程执行中“出现”的所有错误，然后处理它们）。

警告和大多数信息性的注意消息（严重级别小于10或更低的值）不会被TRY...CATCH捕捉，同样，语法和对象名称解析错误也不会被捕捉。虽然这样，这个新的构造仍然是捕捉很多其他常见错误消息的理想选择，而在早期版本中则需要臃肿且笨拙的T-SQL代码来实现。

在通常情况下，你希望确保以某种方式修改数据或参与一个事务的非匿名T-SQL代码的每个块都有一个错误处理程序。然而我不是过分热衷于错误处理的人。我见过一些写代码的人将错误处理放在他们写的每一个SELECT语句中。我个人认为这是过度使用，因为任一导致SELECT语句“中断”的问题都将需要某种人工干预。而且，在.NET错误处理中，在你的SELECT查询中处理错误经常是多余的，因为在应用程序层中可能已经对错误进行处理了。

在下面两个技巧中，演示了两个不同的脚本：一个使用了过时的捕捉错误消息的方法，另一个使用了新的TRY...CATCH语法方法来做同样的事。在这些技巧之后，将演示如何将TRY...CATCH应用到存储过程，以及如何使用嵌套的TRY...CATCH调用。

16.3.1　旧风格的错误处理

在SQL Server 2005之前，错误处理通常需要在每个语句执行之后检查T-SQL @@ERROR函数。然后使

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用GOTO语句跳转到集中的错误处理块，如果错误发生，将终止进程并且回滚事务。

演示如下的代码：

```
DECLARE @ErrorNBR int

BEGIN TRAN

   INSERT Production.Location
   (Name, CostRate, Availability)
   VALUES
   ('Tool Verification', 0.00, 0.00)

   SELECT @ErrorNBR = @@ERROR
   IF @ErrorNBR <> 0
   GOTO UndoTran

   INSERT Production.Location
   (Name, CostRate, Availability)
   VALUES
   ('Frame Forming', 0.00, 0.00)

   SELECT @ErrorNBR = @@ERROR
   IF @ErrorNBR <> 0
   GOTO UndoTran

COMMIT TRAN

UndoTran:
IF @ErrorNBR <> 0
BEGIN
PRINT CAST(@ErrorNBR as varchar(6)) +
' occurred after an attempt to insert into Production.Location'
ROLLBACK TRAN
END
```

这个查询返回：

```
(1行受影响)
消息2601，级别14，状态1，第10行
不能在具有唯一索引'AK_Location_Name' 的对象'Production.Location'中插入重复键的行。
语句已终止。
2601 occurred after an attempt to insert into Production.Location
```

解析

这个技巧中的第一个示例演示了使用SQL Server 2005之前的版本捕捉错误的方法。代码的第一行创建了一个在每个语句执行后保存@@ERRORNBR的整型变量。@@ERRORNBR的值在每个语句执行后都会改变，所以使用局部变量可以保留错误号的原始值：

```
DECLARE @ErrorNBR int
```

接下来，开始事务：

```
BEGIN TRAN
```

对表Production.Location尝试执行两个插入操作。第一个插入的值在表中是不存在的，所以成功执行：

```
INSERT Production.Location
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
(Name, CostRate, Availability)
VALUES
('Tool Verification', 0.00, 0.00)
```

在这个插入操作之后，@@ERROR的值立即被捕捉并保存到@ErrorNBR中：

```
SELECT @ErrorNBR = @@ERROR
```

因为插入成功了，所以值为0。如果插入失败，值等于在sys.messages中适当的错误消息ID。

接下来，IF语句计算局部变量的值，由于它为0，因此它不会调用IF条件：

```
IF @ErrorNBR <> 0
GOTO UndoTran
```

然后尝试另一个插入操作，这次使用在表中存在的位置名称。这次插入操作失败了，原因是对位置名称有唯一约束：

```
INSERT Production.Location
(Name, CostRate, Availability)
VALUES
('Frame Forming', 0.00, 0.00)
```

第一个插入操作的错误捕捉逻辑在第二个插入操作中重复使用，并且在执行时，因为@ErrorNBR不再等于0，所以调用了GOTO部分：

```
SELECT @ErrorNBR = @@ERROR
IF @ErrorNBR <> 0
GOTO UndoTran
```

因为调用了GOTO命令，所以COMMIT TRAN被跳过：

```
COMMIT TRAN
```

UndoTran标签代码打印了错误编号和一条消息，并回滚了事务：

```
UndoTran:
IF @ErrorNBR <> 0
BEGIN
PRINT CAST(@ErrorNBR as varchar(6)) + ' occurred after an
attempt to insert into Production.Location'
ROLLBACK TRAN
END
```

在这个示例中可以明显看到，这个方法需要为每个语句使用重复的代码去捕捉可能发生的错误。对于更大的过程或批处理脚本，为了完成语句级的错误捕捉，这样会显著地增加所需要的T-SQL代码量。

16.3.2 使用 TRY...CATCH 进行错误处理

468

在这个技巧中，将演示相同的错误处理功能，这次使用TRY...CATCH：

```
BEGIN TRY

   BEGIN TRAN

   INSERT Production.Location
   (Name, CostRate, Availability)
   VALUES
   ('Tool Verification', 0.00, 0.00)

   INSERT Production.Location
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    (Name, CostRate, Availability)
   VALUES
   ('Frame Forming', 0.00, 0.00)

   COMMIT TRANSACTION

END TRY
BEGIN CATCH
   SELECT ERROR_NUMBER() ErrorNBR, ERROR_SEVERITY() Severity,
         ERROR_LINE () ErrorLine, ERROR_MESSAGE() Msg

      ROLLBACK TRANSACTION

END CATCH
```

它返回如下的结果：

```
ErrorNBR    Severity    ErrorLine     Msg
2601        14          5             能在具有唯一索引'AK_Location_Name'的对象'Production.Location'中
                                      插入重复键的行。
```

解析

这个技巧的结果与前面技巧的结果相同，只是这次使用了TRY...CATCH。批处理从BEGIN TRY命令开始，并且开始了新的事务：

```
BEGIN TRY

   BEGIN TRAN
```

下一步，再次尝试前面示例中使用的两个插入操作，这次没有在每个语句后使用单独的错误捕捉块：

```
INSERT Production.Location
(Name, CostRate, Availability)
VALUES
('Tool Verification', 0.00, 0.00)

INSERT Production.Location
(Name, CostRate, Availability)
VALUES
('Frame Forming', 0.00, 0.00)
```

TRY批处理包含了希望进行错误检查的语句，使用END TRY关键字结束：

```
END TRY
```

BEGIN CATCH标记了错误处理代码块的开始：

```
BEGIN CATCH
```

使用在这个技巧开始时描述的一些错误函数，报告在TRY块中发生的第一个错误的信息：

```
SELECT   ERROR_NUMBER() ErrorNBR, ERROR_SEVERITY() Severity,
      ERROR_LINE () ErrorLine, ERROR_MESSAGE() Msg
```

然后回滚之前在批处理中声明的打开的事务：

```
ROLLBACK TRANSACTION
```

使用END CATCH命令标记错误处理CATCH块的结束。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

16.3.3　不重编写存储过程来应用 TRY...CATCH

不需要为了从新的TRY...CATCH结构中受益，而去重编写数据库的每个存储过程。你可以使用TRY...CATCH从过程代码外部捕捉和处理错误。

为了演示，下面创建一个执行时会返回错误的存储过程：

```
CREATE PROCEDURE usp_SEL_DivideByZero
AS

SELECT 1/0

GO
```

存储过程没有包含任何错误处理，但如果使用如下的TRY...CATCH，就不会放过任何问题：

```
BEGIN TRY
   EXEC dbo.usp_SEL_DivideByZero
END TRY
BEGIN CATCH
   SELECT ERROR_NUMBER() ErrorNBR, ERROR_SEVERITY() Severity,
         ERROR_LINE () ErrorLine, ERROR_MESSAGE() Msg
   PRINT 'This stored procedure did not execute properly.'
END CATCH
```

它返回：

```
ErrorNBR    Severity    ErrorLine       Msg
8134        16          4               遇到以零作除数错误。
```

解析

尽管在这个示例中创建的存储过程不包括错误处理，但仍然可以通过使用TRY...CATCH执行存储过程来增加对错误的程序级响应。

470

在TRY块中调用过程，并且通过CATCH块捕捉并处理错误信息和消息。

```
BEGIN TRY
   EXEC dbo.usp_SEL_DivideByZero
END TRY
BEGIN CATCH
   SELECT   ERROR_NUMBER() ErrorNBR, ERROR_SEVERITY() Severity,
         ERROR_LINE () ErrorLine, ERROR_MESSAGE() Msg
   PRINT 'This stored procedure did not execute properly.'
END CATCH
```

16.3.4　嵌套错误处理

TRY...CATCH语句可以嵌套使用，这意味着可以在另外的TRY...CATCH块中使用TRY...CATCH语句。甚至在错误处理中也可以使用它来处理可能发生的错误。

在这个示例中，创建一个新的存储过程来处理对HumanResources.Department表的INSERT操作。这个过程包括两层错误处理。如果尝试第一个INSERT操作时发生了错误，则执行第二个尝试，它使用了不同的部门名称：

```
CREATE PROCEDURE dbo.usp_INS_Department
   @Name nvarchar(50),
   @GroupName nvarchar(50)
AS
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
BEGIN TRY

     INSERT HumanResources.Department (Name, GroupName)
    VALUES (@Name, @GroupName)

END TRY
BEGIN CATCH

     BEGIN TRY

     PRINT 'The first department attempt failed.'

     INSERT HumanResources.Department (Name, GroupName)
    VALUES ('Misc', @GroupName)

     END TRY
     BEGIN CATCH
          PRINT 'A Misc department for that group already exists.'
     END CATCH

END CATCH

GO
```

执行既有的Engineering部门的代码导致第一个INSERT操作失败，但第二个关于Research and Development组的Misc部门的INSERT操作成功执行了：

```
EXEC dbo.usp_INS_Department 'Engineering', 'Research and Development'
```

471

它返回：

```
(0行受影响)
The first department attempt failed.
(1行受影响)
```

如果再次尝试这个相同的部门和组的INSERT操作，两个INSERT操作都将失败，导致第二个嵌套的CATCH也返回一个打印的错误：

```
EXEC dbo.usp_INS_Department 'Engineering', 'Research and Development'
```

它返回：

```
(0行受影响)
The first department attempt failed.

(0行受影响)
A Misc department for that group already exists.
```

解析

这个技巧演示了在TRY...CATCH中嵌套另一个TRY...CATCH。使用它可以在错误处理中添加另外的错误处理，在你认为必需的情况下使用它。

浏览这段代码，存储过程前面的几行定义了用于插入到表HumanResources.Department的输入参数：

```
CREATE PROCEDURE dbo.usp_INS_Department
   @Name nvarchar(50),
   @GroupName nvarchar(50)
AS
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接着，第一层TRY块首先尝试向表INSERT新行：

```
BEGIN TRY

    INSERT HumanResources.Department (Name, GroupName)
    VALUES (@Name, @GroupName)

END TRY
```

如果这失败了，CATCH块包含了另一个TRY块：

```
BEGIN CATCH

    BEGIN TRY
```

打印一条语句，然后尝试另一次对表的INSERT操作，这次使用普通的名称Misc，而不是由输入参数传送的原始部门名称：

```
     PRINT 'The first department attempt failed.'

     INSERT HumanResources.Department (Name, GroupName)
    VALUES ('Misc', @GroupName)

    END TRY
```

472

如果这次失败了，嵌套的CATCH将打印另一条消息，告诉用户特定组的Misc部门已经存在：

```
        BEGIN CATCH
            PRINT 'A Misc department for that group already exists.'
        END CATCH

    END CATCH

    GO
```

然后测试存储过程，使用已经存在于表中的部门。因为在部门名称上有UNIQUE约束，所以第一个INSERT操作失败，并且控制权传递到CATCH块。然后CATCH中的TRY块成功地将Misc部门名称插入到表中。

在存储过程的第二次执行中，两个INSERT操作都失败了，但返回了一个PRINT语句，警示你这个事情已被处理了。

473

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第17章 主 体

微软使用一组专门的术语来描述SQL Server的安全功能，它把安全结构分为以下几部分：

- 主体（principal）。这些是可以授予权限以访问特定数据库对象的对象（例如用户登录、角色或应用程序）。
- 安全对象（securable）。这些是访问可以受到控制的对象（例如，表或视图）。
- 权限（permission）。这些是独立权限，可被授予（或阻止）主体以访问安全对象。

主体是本章的主题，安全对象和权限将在下一章中讨论。

主体分为下面3个不同的范围。

- Windows级别的主体是基于Windows域用户账号、域用户组、本地用户账号和本地用户组的主体。一旦添加到SQL Server并给予访问对象的权限，这些类型的主体将基于Windows身份验证获得访问SQL Server的权限。
- SQL Server级别的主体是SQL Server级别的登录名和固定服务器角色。在SQL Server中创建的SQL登录名拥有独立于任何Windows实体的登录名和密码。服务器角色是SQL Server实例级别权限的分组，其他主体能成为它的成员并且继承服务器角色的权限。
- 数据库级别的主体是数据库用户、数据库角色（固定的和用户定义的）和应用程序角色——所有这些内容都将在本章讲到。

我们先来讨论一下Windows级别的主体。

17.1 Windows级别的主体

Windows级别的主体允许使用Windows身份验证访问SQL Server实例。在SQL Server中可以创建基于Windows用户账号或用户组的Windows登录名，它可以属于本地机器或是一个域。Windows登录名可以和域用户、本地用户或Windows用户组进行关联。当添加一个Windows登录名到SQL Server时，用户的名称或用户组会绑定到Windows账号上。添加到SQL Server的Windows登录名不需要独立的密码进行登录，在这种情况下，Windows控制登录身份验证过程。

当用户使用Windows身份验证登录到SQL Server时，它们当前的用户账号必须识别为SQL Server实例的登录名，或者它们必须属于作为一个登录名存在的Windows用户组。

Windows登录名只在服务器操作系统级别应用：不可以将特定数据库对象授予权限到Windows登录名上。要基于Windows登录名授予权限，需要创建一个数据库用户并且将它与这个登录名关联。当我们讲解数据库级别的主体时，将会看到如何完成这个工作。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

安装SQL Server时，要决定是使用Windows身份验证模式还是混合身份验证模式。不管选择哪一个模式，以后你都可以改变主意。由于微软Windows身份验证已集成到Windows操作系统、本地机器和域中，并且不会有密码通过网络传输，所以它要比SQL Server登录名（稍后描述）安全得多。当使用混合身份验证模式时，可以在SQL Server中创建自己的数据库登录名和密码，我们稍后讨论它。

使用CREATE LOGIN命令将Windows用户组或登录名添加到SQL Server实例中。

从Windows用户组或用户登录名创建数据库登录名的部分语法如下：

```
CREATE LOGIN login_name
FROM WINDOWS
[ WITH DEFAULT_DATABASE = database
    | DEFAULT_LANGUAGE = language
]
    | CERTIFICATE certname
    | ASYMMETRIC KEY asym_key_name
```

表17-1描述了这个命令的参数。

表17-1　CREATE LOGIN的参数

参　数	描　述
login_name	这个选项定义了Windows用户或用户组的名称
DEFAULT_DATABASE = database	这个选项指定了Windows登录名的默认数据库上下文，系统数据库master是默认值
DEFAULT_LANGUAGE = language	这个选项指定了Windows登录名的默认语言，如果不指定这个选项，则服务器默认语言为登录名的默认语言
CERTIFICATE certname	这个选项允许绑定一个到Windows登录名的凭证。更多关于凭证的信息请看第19章，完成这个内容的示例请看第20章
ASYMMETRIC KEY asym_key_name	这个选项将密钥绑定到Windows登录名。关于密钥的更多信息请看第19章

17.1.1　创建 Windows 登录名

在这个技巧中，假定在本地机器或域中已经有某个Windows账号和用户组。这个示例在SQL Server实例中创建一个Windows登录名（login），它在内部映射到一个Windows用户：

```
CREATE LOGIN [CAESAR\Livia]
FROM WINDOWS
WITH DEFAULT_DATABASE = AdventureWorks,
DEFAULT_LANGUAGE = English
```

在第二个示例中，创建一个新的基于Windows用户组的Windows登录名。这与前面的示例一样，只是现在映射到一个Windows用户组，而不是一个Windows用户：

476

```
CREATE LOGIN [CAESAR\Senators]
FROM WINDOWS
WITH DEFAULT_DATABASE= AdventureWorks
```

解析

这个技巧演示了添加Windows用户和Windows用户组访问SQL Server实例。在第一个示例中，CREATE LOGIN在中括号中指定了Windows用户：

```
CREATE LOGIN [CAESAR\Livia]
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在下一行中，使用WINDOWS关键字指定新的关联到Windows账号的登录名：

```
FROM WINDOWS
```

下一步，在WITH子句中指定默认数据库和语言：

```
WITH DEFAULT_DATABASE = AdventureWorks,
DEFAULT_LANGUAGE = English
```

在第二个示例中，演示了如何添加Windows用户组到SQL Server，它也需要在CREATE LOGIN命令的中括号中指定：

```
CREATE LOGIN [CAESAR\Senators]
```

FROM WINDOWS子句指定了这是Windows用户组，紧随其后的是默认数据库：

```
FROM WINDOWS
WITH DEFAULT_DATABASE= AdventureWorks
```

当SQL Server登录名关联到Windows用户组时，它使得这个Windows用户组的所有成员继承了Windows登录名的访问权限。所以，这个用户组的所有成员不需要分别显式添加每个Windows账号到SQL Server实例，就拥有了访问SQL Server实例的权限。

17.1.2 查看 Windows 登录名

可以通过查询系统目录视图sys.server_principals查看Windows登录名和用户组。这个示例显示所有可以访问SQL Server的Windows登录名和用户组，以及安全标识符（sid）。每个主体在系统目录视图中都拥有一个sid，它帮助在SQL Server实例中唯一标识出主体：

```
SELECT name, sid
FROM sys.server_principals
WHERE type_desc IN ('WINDOWS_LOGIN', 'WINDOWS_GROUP')
ORDER BY type_desc
```

它返回如下结果（你的结果将会不同）：

```
name                                        sid
BUILTIN\Administrators                      0x0102000000000005200000020020000
CAESAR\SQLServerMSSQLUser$caesar$AUGUSTUS   0x0105000000000005150000019B2983B6D2CB3A
                                            E17D79646EE030000
CAESAR\SQLServerMSFTEUser$CAESAR$AUGUSTUS   0x0105000000000005150000019B2983B6D2CB3A
                                            E17D79646EB030000
CAESAR\SQLServerSQLAgentUser$CAESAR$AUGUSTUS 0x0105000000000005150000019B2983B6D2CB3A
                                            E17D79646EF030000
CAESAR\Senators                             0x0105000000000005150000019B2983B6D2CB3A
                                            E17D79646F1030000
NT AUTHORITY\SYSTEM                         0x0101000000000005120000000
CAESAR\Administrator                        0x0105000000000005150000019B2983B6D2CB3A
                                            E17D79646F4010000
CAESAR\Livia                                0x0105000000000005150000019B2983B6D2CB3A
                                            E17D79646F0030000
```

解析

在这个示例中，演示了如何使用系统目录视图sys.server_principals查询SQL Server实例上的Windows登录名。使用这个视图实际上也可以查看其他主体类型，本章稍后会介绍。

17.1.3 修改 Windows 登录名

一旦添加了Windows登录名到SQL Server中，可以使用ALTER LOGIN命令（这个命令对于SQL登录

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

名会有更多的一些选项，稍后就会看到）来修改它。使用这个命令，你可以完成如下的工作：

- 改变登录名的默认数据库；
- 改变登录名的默认语言；
- 启用或禁用既有的登录名。

部分语法如下所示（参数与CREATE LOGIN相似）：

```
ALTER LOGIN login_name
    {
  ENABLE | DISABLE
    |
WITH
    | DEFAULT_DATABASE = database
    | DEFAULT_LANGUAGE = language }
```

在第一个示例中，在SQL Server中使用的一个Windows登录名（关联到Windows用户的）被禁用。这将禁止这个登录名访问SQL Server，如果已经连接了，就停止它在SQL Server实例上的任何活动：

```
ALTER LOGIN [CAESAR\Livia]
DISABLE
```

下一个示例演示了重新启用这个账号：

```
ALTER LOGIN [CAESAR\Livia]
ENABLE
```

在这个示例中，改变Windows用户组的默认数据库：

```
ALTER LOGIN [CAESAR\Senators]
WITH DEFAULT_DATABASE = master
```

478

解析

在第一个示例中，使用ALTER LOGIN和登录名的名称禁用了一个Windows登录名：

```
ALTER LOGIN [CAESAR\Livia]
```

紧随其后是关键字DISABLE，它禁止了这个账号访问SQL Server实例的权限（它禁止这个账号的访问权限，但是仍将这个登录名保存在SQL Server实例中，以备后面重新启用这个访问权限）：

```
DISABLE
```

第二个示例演示了使用关键字ENABLE重新启用登录名的访问权限。

第三个示例改变了Windows用户组的默认数据库。引用Windows登录名和用户组的语法是相同的，即都是在中括号中指定这两个主体类型：

```
ALTER LOGIN [CAESAR\Senators]
```

然后在第二行为Windows用户组指定了新的默认数据库上下文：

```
WITH DEFAULT_DATABASE = master
```

17.1.4　删除 Windows 登录名

在这个技巧中，将演示使用DROP LOGIN命令从SQL Server实例中完全删除登录名。这个操作会删除登录名访问SQL Server实例的权限。如果这个登录名在被删除时已经连接到SQL Server实例，那么不再允许连接的登录名尝试任何行为。

语法如下：

```
DROP LOGIN login_name
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

唯一的参数是登录名，它可以是Windows或SQL登录名（稍后会演示），正如这个技巧所演示的：

```
-- Windows用户组登录名
DROP LOGIN [CAESAR\Senators]
-- Windows用户登录名
DROP LOGIN [CAESAR\Livia]
```


解析

这个技巧演示了简单的DROP LOGIN命令，它从SQL Server删除登录名。如果登录名拥有任何的安全对象（安全对象的更多信息请看下一章），DROP的尝试将会失败。例如，如果CAESAR\Livia登录名是一个数据库的拥有者，则将引发类似下面的错误：

```
消息15174，级别16，状态1，第3行
登录名' CAESAR\Livia'拥有一个或多个数据库。
在删除该登录名之前，请更改相应数据库的所有者。
```

479

17.1.5 拒绝 Windows 用户或用户组的 SQL Server 访问

使用DENY CONNECT SQL命令来拒绝Windows用户或用户组访问SQL Server。例如：

```
USE [master]
GO
DENY CONNECT SQL TO [CAESAR\Helen]
GO
```

要再允许访问时，你可以使用GRANT命令：

```
USE [master]
GO
GRANT CONNECT SQL TO [CAESAR\Helen]
GO
```

解析

这一部分是对第18章的提前介绍，第18章会更详细地对GRANT和DENY进行描述。简而言之，GRANT命令授予权限到安全对象，DENY拒绝权限到安全对象。使用DENY CONNECT将限制Windows用户或用户组登录名下一次登录时访问SQL Server实例。GRAND CONNECT和DENY CONNECT两者都假定Windows用户或用户组已经在SQL Server存在登录名。要记住你可以拒绝权限存在的限制。例如，如果用如下的代码尝试DENY CONNECT到你自己的登录名：

```
DENY CONNECT SQL TO [CAESAR\Administrator]
```

它返回如下的警告：

```
无法对sa、dbo、实体所有者、information_schema、sys或您自己授予、拒绝或撤销①权限
```

17.2 SQL Server 级别的主体

Windows身份验证依赖于底层的操作系统来完成身份验证（检测某人是哪个特定用户），并且意味着SQL Server要完成必要的授权（决定完成身份验证的用户可以执行什么动作）。当SQL Server主体和SQL Server身份验证一起工作时，SQL Server就自己完成身份验证和授权。

① 在现代汉语词典中，撤销为推荐写法。——编者注

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

正如前面提到过的，当使用混合身份验证模式时，可以在SQL Server中创建自己的登录名的密码。这些SQL登录名只存在于SQL Server中，并且不存在外部Windows用户或用户组的映射。对于SQL登录名，密码存储在SQL Server中。这些用户凭证存储在SQL Server中，并且用来验证用户和决定他的合适的访问权限。

因为这个安全模式涉及显式的密码，所以它没有单独使用Windows身份验证安全。尽管如此，SQL Server登录名仍然广泛地使用在第三方或非Windows操作系统应用程序中。在Windows 2003 Server或更高版本中，SQL Server也已经通过启用类Windows的密码功能增强了密码保护能力，比如强制密码更换、过期时间及其他密码策略（例如密码复杂性）。

与Windows登录名一样，SQL Server登录名也只能用在服务器级别，不可以给它授予权限到特定的数据库对象上。除非你被授予固定服务器角色（比如sysadmin）的成员，否则在使用数据库对象工
480 作之前，必须创建关联到登录名的数据库用户。

正如SQL Server的早期版本，SQL Server支持基于专用登录名和服务器角色的主体，可以将多个专用用户分配到服务器角色。

使用CREATE LOGIN命令创建新的SQL Server登录名：

```
CREATE LOGIN login_name
[WITH PASSWORD = ' password ' [ HASHED ] [ MUST_CHANGE ],
    SID = sid],
    DEFAULT_DATABASE = database,
    DEFAULT_LANGUAGE = language,
    CHECK_EXPIRATION = { ON | OFF},
    CHECK_POLICY = { ON | OFF},
    CREDENTIAL = credential_name ]
```

表17-2中描述了这个命令的参数。

表17-2　CREATE LOGIN的参数

参　数	描　述
login_name	这是登录名
'password' [HASHED] [MUST_CHANGE]	这是登录密码。指定HASHED选项，意思是输入的密码已经经过散列运算（转换到不可读的安全格式）。如果指定MUST_CHANGE，则当用户第一次登录时会提示修改密码
SID = sid	这里显式指定了sid，它将用在SQL Server实例的系统表中。这可以基于来自不同的SQL Server实例的登录名（假如你正在迁移登录名）。如果没有指定它，SQL Server将在系统表中生成自己的sid
DEFAULT_DATABASE = database	这个选项指定SQL登录名的默认数据库上下文，默认值是系统数据库master
DEFAULT_LANGUAGE = language	这个选项指定登录名的默认语言，如果不指定这个选项，登录名的默认语言就是服务器的默认语言
CHECK_EXPIRATION = { ON \| OFF }	当设置为ON（默认值）时，对SQL登录名强制实施密码过期策略。密码过期策略决定密码保持多久有效之后就必须改变。这个功能需要Windows 2003 Server或更高版本的支持
CHECK_POLICY = { ON \| OFF }	当设置为ON（默认值）时，Windows密码策略应用到SQL登录名上（例如，关于密码长度、复杂性和包括非字母数字字符的策略）。这个功能需要Windows 2003 Server或更高版本的支持
481 CREDENTIAL = credential_name	这个选项允许服务器凭证映射到SQL登录名。更多关于凭证的信息请看第18章

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

17.2.1 创建 SQL Server 登录名

这个示例首先演示了如何创建带密码并指定了默认数据库的SQL Server登录名：

```
CREATE LOGIN Veronica
WITH PASSWORD = 'InfernoII',
DEFAULT_DATABASE = AdventureWorks
```

假定使用Windows 2003 Server或更高版本，并且是混合身份验证方式，技巧继续创建一个SQL登录名，它的密码在用户第一次登录时必须要修改。这个登录名也通过CHECK_POLICY选项ON被创建，需要它遵从Windows密码策略：

```
CREATE LOGIN Trishelle
WITH PASSWORD = 'ChangeMe' MUST_CHANGE ,
     CHECK_EXPIRATION = ON,
        CHECK_POLICY = ON
```

解析

在这个技巧的第一个示例中演示了创建名为Veronica的SQL登录名。在CREATE LOGIN后指定了登录名的名称：

```
CREATE LOGIN Veronica
```

第二行指定了登录名的密码：

```
WITH PASSWORD = 'InfernoII'
```

代码的最后一行指定了登录名上下文在登录到SQL Server之后会首先进入的默认数据库：

```
DEFAULT_DATABASE = AdventureWorks
```

第二个SQL登录名示例演示了如何通过在密码之后指定MUST_CHANGE标识，强制在第一次登录时修改密码：

```
CREATE LOGIN Trishelle
WITH PASSWORD = 'ChangeMe' MUST_CHANGE ,
```

这个密码策略的集成需要Windows 2003 Server的支持，也为这个登录名指定了密码过期时间和密码策略选项：

```
CHECK_EXPIRATION = ON,
CHECK_POLICY = ON
```

17.2.2 查看 SQL Server 登录名

可以通过查询系统目录视图sys.server_principals来查看SQL Server登录名（以及其他主体）：

```
SELECT name, sid
FROM sys.server_principals
WHERE type_desc IN ('SQL_LOGIN')
ORDER BY name
```

它返回如下的结果：

```
name        sid
Boris       0xC2692B07894DFD45913C5595C87936B9
BrianG      0x4EC3966D4E33844F89680AFD87D2D5BD
JoeSa       0xB64D3C39533CC648B581884EC143F2D4
Prageeta    0x00CACEF1F0E0CE429B7C808B11A624E7
sa          0x01
SteveP      0xAAA2CD258750C641BBE9584627CAA11F
Veronica    0xE08E462A75D8C047A4561D4E9292296D
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

这个技巧的查询通过查询sys.server_principals目录视图返回了SQL Server实例上的所有SQL登录名的名称和sid。

17.2.3 修改 SQL Server 登录名

一旦登录名添加到SQL Server中，就可以使用ALTER LOGIN命令来修改它。使用这个命令，可以完成如下一些任务：

- 修改登录名的密码；
- 修改默认数据库或语言；
- 不破坏登录名当前分配的权限，修改既有的登录名的名称；
- 修改密码策略设置（启用或禁用它们）；
- 从一个SQL登录名凭证映射或删除映射；
- 启用或禁用既有的登录名；
- 给锁定的登录名解锁。

语法参数与CREATE LOGIN相似（我会在这个技巧中演示使用方法）：

```
ALTER LOGIN login_name
    {
   ENABLE | DISABLE
    |
WITH PASSWORD = ' password '
    [ OLD_PASSWORD = ' oldpassword '
      | [ MUST_CHANGE | UNLOCK ] ]
    | DEFAULT_DATABASE = database
    | DEFAULT_LANGUAGE = language
    | NAME = login_name
    | CHECK_POLICY = { ON | OFF }
    | CHECK_EXPIRATION = { ON | OFF }
    | CREDENTIAL = credential_name
    | NO CREDENTIAL
    }
```

在这个技巧的第一个示例中，SQL登录名的密码被从InfernoII修改为InfernoIII：

```
ALTER LOGIN Veronica
WITH PASSWORD = 'InfernoIII'
OLD_PASSWORD = 'InfernoII'
```

483

OLD_PASSWORD是当前要修改的密码，如果要修改固定服务器角色sysadmin的成员的密码，就不需要知道旧密码。

第二个示例演示了修改SQL登录名Veronica的默认数据库：

```
ALTER LOGIN Veronica
WITH DEFAULT_DATABASE = [AdventureWorks]
```

这个技巧的第三个示例演示了同时修改SQL登录名的名称和密码：

```
ALTER LOGIN Veronica
WITH NAME = Angela,
PASSWORD = 'BOS2004'
```

与删除并创建新的登录名相比，修改登录名的名称带来的一个主要的好处是——当登录名被重命名时，分配到原始登录名的权限不会被破坏。在这个例子中，登录名Veronica重命名为Angela，但是

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

权限仍然是一样的。

解析

在这个技巧的第一个示例中，使用ALTER LOGIN指定旧密码和新密码去修改密码。如果你拥有固定服务器角色sysadmin权限，那么你只需要指定新密码。第二个示例演示了如何修改SQL登录名的默认数据库。最后一个示例演示了如何修改登录名的名称，将Veronica修改为Angela，同时修改登录名的密码。

17.2.4 管理登录名的密码

SQL Server提供了LOGINPROPERTY函数返回登录名和密码策略设置和状态信息，可以用来确定SQL登录名的如下属性：

- 登录名是否锁定或过期；
- 登录名的密码是否必须修改；
- 密码重试次数和最后输入不正确密码的时间；
- 登录超时时间；
- 最近设置密码的时间及使用密码策略跟踪登录的时间长度；
- 迁移时使用的密码散列值（例如，迁移到另一个SQL实例）。

这个函数有两个参数：SQL登录名的名称和要检查的属性。在这个示例中，将列出指定登录名所有可用属性：

```
SELECT LOGINPROPERTY('Angela', 'IsLocked') IsLocked,
       LOGINPROPERTY('Angela', 'IsExpired') IsExpired,
       LOGINPROPERTY('Angela', 'IsMustChange') IsMustChange,
       LOGINPROPERTY('Angela', 'BadPasswordCount') BadPasswordCount,
       LOGINPROPERTY('Angela', 'BadPasswordTime') BadPasswordTime,
       LOGINPROPERTY('Angela', 'HistoryLength') HistoryLength,
       LOGINPROPERTY('Angela', 'LockoutTime') LockoutTime,
       LOGINPROPERTY('Angela', 'PasswordLastSetTime') PasswordLastSetTime,
       LOGINPROPERTY('Angela', 'PasswordHash') PasswordHash
```

这个查询返回：

IsLocked	IsExpired	IsMustChange	BadPassword Count	BadPassword Time	HistoryLength	LockoutTime	PasswordLast SetTime	PasswordHash
0	0	0	0	1900-01-01 00:00:00.000	0	1900-01-01 00:00:00.000	2007-12-22 07:07:33.590	0x01000D175F71 610D24501843F3 F3E08E518B8DF7 73E9006C4DFA

解析

使用LOGINPROPERTY可以验证SQL登录名的属性。可以使用它来管理密码的重设，例如，检查最后设置密码的时间，然后修改所有在某一时间范围内没有更新过密码的登录名。

还可以让密码散列值属性与CREATE LOGIN和hashed_password HASHED参数一起使用，在新的SQL Server实例中以保留的密码重新创建SQL登录名。

17.2.5 删除 SQL 登录名

这个技巧演示了从SQL Server实例中通过使用DROP LOGIN命令删除SQL登录名。

语法如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DROP LOGIN login_name
```

唯一的参数是登录名名称——它可以为Windows或SQL登录名，正如这个技巧所演示的那样：

```
DROP LOGIN Angela
```

解析

这个技巧演示了简单的DROP LOGIN命令，它从SQL Server中删除登录名。处理过程是简单的，但是如果登录名拥有任何安全对象（关于安全对象的信息请看下一章），DROP的尝试将会失败。例如，如果Angela登录名是一个数据库的拥有者，将会引发如下类似的错误信息：

```
消息15174，级别16，状态1，第1行
登录名'Angela' 拥有一个或多个数据库。在删除该登录名之前，请更改相应数据库的所有者。
```

17.2.6 管理服务器角色成员

固定服务器角色是预定义的SQL用户组，它们被赋予特定的SQL Server范围（与数据库或架构范围相对）的权限。不可以创建新的固定服务器角色，只可以从其他SQL或Windows登录名添加或删除成员到这个角色中。

固定服务器角色sysadmin是SQL Server中拥有最高级别权限的角色。尽管服务器角色是基于权限的，但是它们拥有成员（SQL或Windows登录名 / 用户组）并且被微软分类为主体。

485 使用系统存储过程sp_addsrvrolemember来添加登录名到固定服务器角色中。

语法如下：

```
sp_addsrvrolemember [ @loginame= ] 'login',
[ @rolename = ] 'role'
```

系统存储过程的第一个参数是添加到固定服务器角色的登录名。第二个参数是要添加登录名的固定服务器角色。

在这个示例中，创建了登录名Veronica，然后把它添加到固定服务器角色sysadmin中：

```
CREATE LOGIN Veronica
WITH PASSWORD = 'PalmTree1'
GO

EXEC master..sp_addsrvrolemember
'Veronica',
'sysadmin'
GO
```

使用系统存储过程sp_dropsrvrolemember来从固定服务器角色删除登录名。语法几乎与sp_addsrvrolemember相同：

```
sp_dropsrvrolemember [ @loginame= ] 'login' ,
[ @rolename= ] 'role'
```

这个示例从固定服务器角色中删除登录名Veronica：

```
EXEC master..sp_dropsrvrolemember
'Veronica',
'sysadmin'
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

一旦将登录名添加到固定服务器角色中，登录名就会接受与这个固定服务器角色关联的权限。使用系统存储过程sp_addsrvrolemember来将新的登录名添加到一个固定角色成员，并使用sp_dropsrvrolemember来从固定角色成员中删除登录名。

将SQL或Windows登录名添加到固定服务器角色不是一个轻松的工作。固定服务器角色包含了广泛的权限——作为最理想的规则，仅为要进行的工作授予那些绝对必要的权限。例如，不要将某些只需要一个表的SELECT权限的人放置到sysadmin成员中。

17.2.7 报告固定服务器角色信息

固定服务器角色定义成组的SQL Server范围的权限（比如备份数据库和创建新的登录名）。和SQL或Windows登录名相似，固定服务器角色拥有一个安全标识符（sid），并且可以在系统目录视图sys.server_principals中进行查看。和SQL或Windows登录名不同的是，固定服务器角色可以拥有在其中定义的成员（SQL和Windows登录名），这些成员继承固定服务器角色的权限。

查询系统目录视图sys.server_principals来查看固定服务器角色：

```
SELECT name
FROM sys.server_principals
WHERE type_desc = 'SERVER_ROLE'
```

这个查询返回：

486

```
name
public
sysadmin
securityadmin
serveradmin
setupadmin
processadmin
diskadmin
dbcreator
bulkadmin
```

也可以通过执行系统存储过程sp_helpserverrole来查询固定服务器角色列表：

```
EXEC sp_helpsrvrole
```

它返回：

```
ServerRole          Description
sysadmin            System Administrators
securityadmin       Security Administrators
serveradmin         Server Administrators
setupadmin          Setup Administrators
processadmin        Process Administrators
diskadmin           Disk Administrators
dbcreator           Database Creators
bulkadmin           Bulk Insert Administrators

(8 行受影响)
```

表17-3详细描述了授予每个固定服务器角色的权限。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表17-3　服务器角色权限

服务器角色	授予的权限
sysadmin	GRANT选项（可以GRANT权限给其他人），CONTROL SERVER
setupadmin	ALTER ANY LINKED SERVER
serveradmin	ALTER SETTINGS、SHUTDOWN、CREATE ENDPOINT、ALTER SERVER STATE、ALTER ANY ENDPOINT、ALTER RESOURCES
securityadmin	ALTER ANY LOGIN
processadmin	ALTER SERVER STATE、ALTER ANY CONNECTION
diskadmin	ALTER RESOURCES
dbcreator	CREATE DATABASE
bulkadmin	ADMINISTER BULK OPERATIONS

可以执行系统存储过程sp_helpsrvrolemember来查看某个固定服务器角色的成员：

487

```
EXEC sp_helpsrvrolemember 'sysadmin'
```

它返回如下的结果：

```
ServerRole      MemberName                                  MemberSID
sysadmin        sa                                          0x01
sysadmin        NT AUTHORITY\SYSTEM                         0x0101000000000000512000000
sysadmin        BUILTIN\Administrators                      0x0102000000000005200000002
                                                            0020000
sysadmin        CAESAR\SQLServerMSSQLUser$caesar$AUGUSTUS   0x0105000000000005150000001
                                                            9B2983B6D2CB3AE17D79646EE0300
                                                            00
sysadmin        CAESAR\SQLServerMSFTEUser$CAESAR$AUGUSTUS   0x0105000000000005150000001
                                                            9B2983B6D2CB3AE17D79646EB0300
                                                            00
sysadmin        CAESAR\Administrator                        0x0105000000000005150000001
                                                            9B2983B6D2CB3AE17D79646F40100
                                                            00
sysadmin        CAESAR\SQLServerSQLAgentUser$CAESAR$AUGUSTUS 0x0105000000000005150000001
                                                            9B2983B6D2CB3AE17D79646EF0300
                                                            00
```

解析

可以查询系统目录视图sys.server_principals来查看固定服务器角色，或者可以使用系统存储过程sp_helpsrvrole来查看所有角色的描述。使用系统存储过程sp_helpsrvrolemember来查看一个角色（其他主体）的成员。

下一技巧将展示如何从固定服务器角色中添加或删除其他主体。

17.3　数据库级别的主体

数据库级别的主体是表示可以分配访问数据库或数据库中的特殊对象的权限给用户的对象。登录名在服务器级别操作，用它可以执行比如像连接到SQL Server的活动，而数据库主体在数据库级别操作，用它可以查询或管理数据、对数据库中的对象执行DDL语句或在数据库级别管理用户的权限。

SQL Server中有下面3种不同类型的数据库主体。

- 数据库用户。数据库用户主体是执行数据库内的请求的数据库级别的安全上下文，并且与SQL Server或Windows登录名进行关联。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 数据库角色。数据库角色又分为两种，固定的和用户定义的。固定数据库角色可以在SQL Server实例的所有数据库中找到，并且拥有分配给它们的数据库作用域的权限（比如对所有表的SELECT权限或CREATE表的能力）。用户定义数据库角色是可以自行创建的，并允许你管理安全对象上的权限，这要比给多个数据库用户单独授予相似的权限更加容易。
- 应用程序角色。应用程序角色是不包含成员的成组的权限。但是，能够以应用程序角色的身份“登录”。当使用一个应用程序角色时，它会覆盖登录名的其他所有权限，只授予你应用程序角色的那些权限。

488

在这一部分中，将介绍如何创建、修改、报告和删除数据库用户。还将介绍如何使用数据库角色（固定的和用户定义的）以及应用程序角色。

17.3.1 创建数据库用户

一旦创建了登录名，就可以把它映射到数据库用户。一个登录名可以映射到一个SQL Server实例的多个数据库上，但是每一个数据库只有一个用户能访问。

使用CREATE USER命令为用户授权访问。

语法如下：

```
CREATE USER user_name
    [ FOR
        { LOGIN login_name
        | CERTIFICATE cert_name
        | ASYMMETRIC KEY asym_key_name
        }
    ]
    [ WITH DEFAULT_SCHEMA = schema_name ]
```

表17-4描述了这个命令的参数。

表17-4 CREATE USER参数

参 数	描 述
user_name	这个参数定义了数据库中用户的名称
login_name	这个参数定义了映射到数据库用户的SQL或Windows登录名的名称
cert_name	当指定该选项时，它指定绑定到数据库用户的凭证。更多关于凭证的信息请看第19章
asym_key_name	当指定该选项时，它指定绑定到数据库用户的非对称密钥。更多关于密钥的信息请看第19章
schema_name	这个参数表示用户将属于的默认架构，当用户引用数据库对象时它将决定首先要检查的架构。如果未指定这个选项，则将使用dbo架构，对于一个尚未在数据库中创建的架构也可以指定该架构名

在技巧的第一个示例中，在数据库TestDB中创建名为Veronica的新用户：

```
IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'TestDB')
BEGIN
   CREATE DATABASE TestDB
END
GO

USE TestDB
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE USER Veronica
```

489 在第二个示例中，指定了默认架构且名为Joe的数据库用户映射到Windows登录名：

```
USE TestDB
GO

CREATE USER Helen
FOR LOGIN [CAESAR\Helen]
WITH DEFAULT_SCHEMA = HumanResources
```

解析

在技巧的第一个示例中，在数据库TestDB中创建了一个名为Veronica的用户。如果不在CREATE USER中指定FOR LOGIN子句，它会尝试将用户映射到相同名称的登录名上（在这个例子中，登录名是Veronica）。注意这里没有指定默认架构，这意味着Veronica的默认架构将为dbo。

在第二个示例中，在数据库AdventureWorks中创建了名为Helen的新用户，它映射到名称为[CAESAR\Helen]（注意要使用中括号）的Windows登录名上。Helen登录名的默认架构设置为HumanResources。对于Helen登录名使用的所有未限定对象引用的查询，SQL Server将首先在HumanResources架构中查找对象。

17.3.2　报告数据库用户信息

可以通过使用系统存储过程sp_helpuser来报告当前数据库连接的数据库用户（和角色）信息。

语法如下：

```
sp_helpuser [ [ @name_in_db= ] ' security_account ' ]
```

那个可选的参数是希望返回信息的数据库用户的名称。例如：

```
EXEC sp_helpuser 'Veronica'
```

这个查询返回如下结果：

UserName	RoleName	LoginName	DefDBName	DefSchemaName	UserID	SID
Veronica	public	Veronica	master	dbo	5	0x3057F4EEC4F07A46BF126AB2434F104D

解析

系统存储过程sp_helpuser返回在当前数据库中定义的数据库用户。从结果中可以确定用户名称、登录名称、默认数据库和架构以及用户的安全标识符（sid）等重要信息。如果没有指定具体的用户，sp_helpuser将返回连接到的当前数据库的所有用户信息。

17.3.3　修改数据库用户

可以通过使用ALTER USER命令来重命名数据库用户或修改用户的默认架构。

语法如下：

```
ALTER USER user_name
     WITH NAME = new_user_name
     | DEFAULT_SCHEMA = schema_name
```

490 在这个技巧的第一个示例中，改变了数据库用户Helen的默认架构：

```
USE TestDB
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ALTER USER Helen
WITH DEFAULT_SCHEMA = Production
```

在这个技巧的第二个示例中，修改了数据库用户名称：

```
USE TestDB
GO

ALTER USER Veronica
WITH NAME = VSanders
```

解析

ALTER USER命令允许你实现下面的某个修改：重命名数据库用户或修改数据库用户的默认架构。第一个示例将Helen登录名的默认架构修改为架构Production。第二个示例将数据库用户Veronica重命名为VSanders。

17.3.4 从数据库删除数据库用户

使用DROP USER命令从数据库中删除用户。

语法如下：

```
DROP USER user_name
```

user_name是数据库用户的名称，如这个示例所演示的：

```
USE TestDB
GO

DROP USER VSanders
```

解析

DROP USER命令从数据库中删除用户，但不会影响关联到它的Windows或SQL登录名。像DROP LOGIN一样，不可以删除作为数据库对象的拥有者的用户。例如，如果数据库用户Helen是名为Test架构的架构拥有者，将得到类似下面的错误：

```
消息15138，级别16，状态1，第2行
数据库主体在该数据库中拥有架构，无法删除。
```

17.3.5 修复孤立的数据库用户

当数据库迁移到新的服务器时（例如通过使用BACKUP/RESTORE），登录名和数据库用户的关系可能会被破坏。登录名拥有一个安全标识符，它在SQL Server实例中唯一标识出这个登录名。这个sid用于保存登录名在每个能访问的数据库中关联的数据库用户。使用相同的名称在不同的SQL Server实例中创建另外的登录名不会重新创建相同的sid，除非在CREATE LOGIN语句的sid参数中特别指定它。

491

下面的查询通过在列sid上联结系统目录视图sys.database_principals和目录视图sys.server_principals，演示了这个连接来查找数据库中孤立的用户：

```
SELECT dp.name OrphanUser, dp.sid OrphanSid
FROM sys.database_principals dp
LEFT OUTER JOIN sys.server_principals sp ON
   dp.sid = sp.sid
WHERE sp.sid IS NULL AND
      dp.type_desc = 'SQL_USER' AND
      dp.principal_id > 4
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回：

```
OrphanUser      OrphanSid
Sonja           0x40C455005F34E44FB95622488AF48F75
```

如果从不同的SQL Server实例到新的SQL Server实例RESTORE数据库，并且在新的SQL Server实例上数据库用户没有关联的登录名，数据库用户可能就会变成“孤立的”。如果在新的SQL Server实例上存在和数据库用户名称相匹配的相同名称的登录名，但是登录名的sid与恢复的数据库用户sid不匹配，那么数据库中的数据库用户仍然可能是孤立的。

从SQL Server的前一个版本（SQL Server 2005 Service Pack 2）开始，就可以使用ALTER USER WITH LOGIN命令来重映射登录名/用户的关联。它可以应用于SQL和Windows账户，如果在活动目录中重建底层Windows用户或用户组，并且现在它们的标识不再映射到SQL Server实例产生的sid上时是非常有用的。

下面的查询演示了重映射孤立数据库用户Sonja到关联的服务器登录名上：

```
ALTER USER Sonja
WITH LOGIN = Sonja
```

下一个示例演示映射数据库用户[Hellen]到登录名[CAESAR\Helen]（假设因为删除或重建了外部的SQL Server，用户与Windows账户孤立或域账户的sid发生了变更）：

```
ALTER USER [Helen]
WITH LOGIN = [CAESAR\Helen]
```

这个命令还可以用在将用户重映射到新的登录名上，不管用户是否是孤立的。

解析

在这个技巧中，演示了查询sys.database_principals和sys.server_principals目录视图来查看所有sid不存在于服务器范围（没有关联的登录名sid）的数据库用户。然后演示了使用ALTER USER将数据库用户映射到同名的登录名上（但sid不同）。还演示了如何使用ALTER USER重映射孤立的Windows账户。

492

提示 在SQL Server早期的版本中，可以使用存储过程sp_change_users_login来执行并报告sid的重映射。这个存储过程不再建议使用，请用ALTER USER WITH LOGIN来替代。

17.3.6 报告固定数据库角色信息

固定数据库角色能在SQL Server实例的每个数据库中找到，并且它们具有分配给它们的数据库范围的权限（比如所有表的SELECT权限或CREATE表的能力）。和固定服务器角色类似，固定数据库角色拥有成员（数据库用户），这些成员继承这个角色的权限。

可以通过执行系统存储过程sp_helpdbfixedrole来查看固定数据库角色的列表：

```
EXEC sp_helpdbfixedrole
```

它返回如下的结果：

```
DBFixedRole         Description
db_owner            DB Owners
db_accessadmin      DB Access Administrators
db_securityadmin    DB Security Administrators
db_ddladmin         DB DDL Administrators
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
db_backupoperator       DB Backup Operator
db_datareader           DB Data Reader
db_datawriter           DB Data Writer
db_denydatareader       DB Deny Data Reader
db_denydatawriter       DB Deny Data Writer
```


可以执行系统存储过程sp_helprolemember来查看固定数据库角色（任何用户定义的角色或应用程序角色）的数据库成员。

```
EXEC sp_helprolemember
```

它返回如下的结果（成员sid引用了映射到数据库用户的登录名的sid）：

```
DbRole              MemberName      MemberSid
db_backupoperator   Joe             0x0105000000000005150000000527A777BF094B3850F
db_datawriter       Joe             0x0105000000000005150000000527A777BF094B3850FF83D0
db_owner            dbo             0x01
```

解析

固定数据库角色能在SQL Server实例的每个数据库中找到。一个固定数据库角色将重要的数据库权限汇集到一起。这些权限不可以修改或删除。

在这个技巧中，执行存储过程sp_helpdbfixedrole列出可获得的固定数据库角色：

```
EXEC sp_helpdbfixedrole
```


在这之后，使用系统存储过程sp_helprolemember列出所有固定数据库角色的成员（数据库用户），显示角色的名称、数据库用户名称以及登录名sid：

```
EXEC sp_helprolemember
```

像固定服务器角色一样，对于数据库用户，最好不要在没有确认所有权限都是绝对必要的情况下，将它授予到固定数据库角色的成员中。例如，不要在用户只需要一个表的SELECT权限时授予它db_owner成员关系。

下一个技巧将展示如何从固定数据库角色中添加或删除数据库用户。

17.3.7 管理固定数据库角色成员

要将一个数据库用户或角色与一个数据库角色（用户定义角色或应用程序角色）相关联，可以使用系统存储过程sp_addrolemember来实现。

语法如下：

```
sp_addrolemember [ @rolename = ] 'role',
    [ @membername = ] 'security_account'
```

系统存储过程sp_addrolemember的第一个参数指定了角色的名称，第二个参数是数据库用户的名称。

要删除数据库用户和角色之间的关系，可以使用系统存储过程sp_droprolemember来实现：

```
sp_droprolemember [ @rolename= ] 'role' ,
          [ @membername= ] 'security_account'
```

像系统存储过程sp_addrolemember一样，系统存储过程sp_droprolemember的第一个参数指定了角色的名称，第二个参数是数据库用户的名称。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第一个示例演示了向固定角色db_datawriter和db_datareader添加数据库用户Veronica：

```
USE TestDB
GO
EXEC sp_addrolemember 'db_datawriter', 'Helen'
EXEC sp_addrolemember 'db_datareader', 'Helen'
```

第二个示例演示了如何从角色db_datawriter中删除数据库用户Helen：

```
USE TestDB
GO
EXEC sp_droprolemember 'db_datawriter', 'Helen'
```

解析

这个技巧先讨论系统存储过程sp_addrolemember，使用它可以添加一个数据库用户到既有的数据库角色中，数据库用户Helen被添加到了db_datawriter和db_datareader中，这会给予这个用户对数据库AdventureWorks中所有表或视图的SELECT、INSERT、UPDATE或DELETE所有这些权限。

```
EXEC sp_addrolemember 'db_datawriter', 'Helen'
EXEC sp_addrolemember 'db_datareader', 'Helen'
```

存储过程的第一个参数是数据库角色，第二个参数是关联到数据库角色的数据库用户（或角色）
494 的名称。

在这之后，使用sp_droprolemember从角色db_datawriter中删除Helen的成员关系。

```
EXEC sp_droprolemember 'db_datawriter', 'Helen'
```

17.3.8　管理用户定义的数据库角色

用户定义的数据库角色可以让你更容易地管理到安全对象的权限，使用它就不再需要反复地分别给多个数据库用户授予相同的权限。你可以创建一个数据库角色，授予它到安全对象的权限，然后添加一个或多个数据库用户到数据库角色作为它的成员。当需要修改权限时，只需要修改一个数据库角色的权限，然后角色的成员将自动继承这些权限的改变。

使用CREATE ROLE命令在数据库中创建用户定义角色。

语法如下：

```
CREATE ROLE role_name [ AUTHORIZATION owner_name ]
```

这个命令指定了新角色的名称，以及可选的角色拥有者名称。拥有者名称是拥有新数据库角色的用户或数据库角色的名称（从而就可以管理它了）。

可以通过执行sp_helprole系统存储过程列出所有的数据库角色（固定的、用户定义的和应用程序角色）。

```
EXEC sp_helprole
```

它返回如下部分结果（列IsAppRole为1代表角色是应用程序角色，0则不是）：

```
RoleName        RoleId          IsAppRole
public          0               0
db_owner        16384           0
...             ...             ...
```

一旦在数据库中创建了数据库角色，就像操作普通的数据库用户一样，可以授予它或拒绝给它权限（更多关于权限的内容请看下一章）。马上会演示将权限授予数据库角色。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果希望改变数据库角色的名称，但不破坏角色当前的权限和成员关系，可以使用ALTER ROLE命令，语法如下：

```
ALTER ROLE role_name WITH NAME = new_name
```

该命令把原始角色的名称作为第一个参数，新角色名称为第二个参数。

使用DROP ROLE命令删除角色，语法如下：

```
DROP ROLE role_name
```

如果角色拥有任何安全对象，在可以删除角色之前，需要把角色关系转移到新的拥有者上。

在这个例子中，在AdventureWorks数据库中创建一个新的角色。

```
USE AdventureWorks
GO
CREATE ROLE HR_ReportSpecialist AUTHORIZATION db_owner
```

在创建了角色之后，新角色还没有拥有任何数据库权限。在下一个查询中，将对表HumanResources.Employee的SELECT权限授予数据库角色HR_ReportSpecialist：

```
GRANT SELECT ON HumanResources.Employee TO HR_ReportSpecialist
```

要将数据库用户Veronica添加到一个新的数据库角色中，可以使用系统存储过程sp_addrolemember来实现：

```
EXEC sp_addrolemember 'HR_ReportSpecialist',
          'Veronica'
GO
```

如果后来我们发现角色的名称和它的用途不匹配，可以使用ALTER ROLE改变它的名称：

```
ALTER ROLE HR_ReportSpecialist WITH NAME = HumanResources_RS
```

即使改变了角色的名称，Veronica仍为角色的成员。

最后一个示例演示了如何删除数据库角色：

```
DROP ROLE HumanResources_RS
```

它返回一个错误消息，因为只有角色中没有成员才能删除：

```
消息15144，级别16，状态1，第1行
该角色有多个成员，角色必须为空后才能删除。
```

所以，在删除这个角色之前，先删除这个角色中唯一的成员：

```
EXEC sp_droprolemember 'HumanResources_RS',
          'Veronica'
GO

DROP ROLE HumanResources_RS
```

解析

CREATE ROLE命令在数据库中创建了新的数据库角色。在创建之后，就像操作普通的数据库用户一样，可以附加权限到角色上。角色允许在组级别上管理权限——允许每个角色成员以一致的方式继承权限，而不是附加权限到每一个用户上，这可能相同或可能不同。

这个技巧演示了一些与管理用户定义数据库角色相关的命令。使用系统存储过程sp_helprole列出了当前数据库中所有的数据库角色。使用CREATE ROLE创建了固定数据库角色db_owner拥有的新的用户定义角色：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE ROLE HR_ReportSpecialist AUTHORIZATION db_owner
```

然后我授予一个表的SELECT权限给新角色：

```
GRANT SELECT ON HumanResources.Employee TO HR_ReportSpecialist
```

然后将用户Veronica添加到新角色中：

```
EXEC sp_addrolemember 'HR_ReportSpecialist',
        'Veronica'
```

496 接着使用ALTER ROLE修改角色的名称（仍然完整地保留成员关系和权限）：

```
ALTER ROLE HR_ReportSpecialist WITH NAME = HumanResources_RS
```

然后从角色中删除用户Veronica（这样我们才可以删除用户定义角色）：

```
EXEC sp_droprolemember 'HumanResources_RS',
         'Veronica'
```

一旦清空了成员，就可以删除用户定义数据库角色了：

```
DROP ROLE HumanResources_RS
```

17.3.9　管理应用程序角色

应用程序角色是由登录名和数据库角色混合而成的。能够以与分配给用户定义角色权限相同的方式，分配权限给应用程序角色。应用程序角色与数据库和服务器角色不同，在应用程序角色中不允许拥有成员。取而代之的是，应用程序角色被使用启用密码的系统存储过程激活（activated）。当使用应用程序角色时，它将覆盖登录名可能拥有的所有其他权限。

因为应用程序角色不包含成员，为了启用权限，它需要密码。除此之外，一旦会话的上下文设置为使用应用程序角色，所有既有的用户或登录名权限都将无效，只有应用程序角色的权限有效。

使用CREATE APPLICATION ROLE创建应用程序角色，语法如下：

```
CREATE APPLICATION ROLE application_role_name
    WITH PASSWORD = ' password ' [ , DEFAULT_SCHEMA = schema_name ]
```

表17-5描述了这个命令的参数。

表17-5　CREATE APPLICATION ROLE的参数

参　数	描　述
application_role_name	应用程序角色的名称
password	激活应用程序角色访问权限的密码
schema_name	应用程序角色的默认数据库架构。它是对于查询中未受限的对象名要检查的架构

在这个示例中，创建了新的应用程序角色名称DataWareHouseApp，并且授予它查看数据库AdventureWorks中视图的权限：

```
USE AdventureWorks
GO
CREATE APPLICATION ROLE DataWareHouseApp
WITH PASSWORD = 'mywarehouse123!',
 DEFAULT_SCHEMA = dbo
```

在没有先授予应用程序角色能做一些事情的权限时，它本身是没有用的。所以，在这个示例中，将对指定数据库视图的SELECT权限授予应用程序角色：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
-- 现在授予这个应用程序权限
GRANT SELECT ON Sales.vSalesPersonSalesByFiscalYears
TO DataWareHouseApp
```

使用系统存储过程sp_setapprole激活当前用户会话的应用程序角色的权限。在下一个示例中，我激活一个应用程序角色并查询两个表：

```
EXEC sp_setapprole 'DataWareHouseApp', -- 应用程序角色的名称。
'mywarehouse123!' -- 密码
-- 工作
SELECT COUNT(*)
FROM Sales.vSalesPersonSalesByFiscalYears
-- 不工作
SELECT COUNT(*)
FROM HumanResources.vJobCandidate
```

它返回：

```
-----------
14

(1行受影响)

消息229，级别14，状态5，第7行
拒绝了对对象'vJobCandidate' (数据库'AdventureWorks'，架构'HumanResources')的SELECT权限。
```

即使原始连接的登录名是拥有sysadmin权限的一个登录名，使用sp_setapprole进入到应用程序权限也意味着只应用这个角色的权限。因此，在这个例子中，应用程序角色拥有对视图Sales.vSalesPersonSalesByFiscalYears的SELECT权限，却没有查询视图HumanResources.vJobCandidate的权限。

为了恢复到原始登录名的权限，必须关闭连接并打开新的连接。

可以使用ALTER APPLICATION ROLE命令修改应用程序角色的名称、密码或默认数据库。

语法如下：

```
ALTER APPLICATION ROLE application_role_name
WITH NAME = new_application_role_name
    | PASSWORD = ' password '
    | DEFAULT_SCHEMA = schema_name
```

表17-6描述了这个命令的参数。

表17-6 ALTER APPLICATION ROLE的参数

参 数	描 述
new_application_role_name	新应用程序角色的名称
password	新应用程序角色的密码
schema_name	新默认架构

在这个示例中，改变了应用程序角色的名称和密码：

```
ALTER APPLICATION ROLE DataWareHouseApp
WITH NAME = DW_App, PASSWORD = 'newsecret!123'
```

使用DROP APPLICATION ROLE从数据库中删除应用程序角色，语法如下：

```
DROP APPLICATION ROLE rolename
```

这个命令只有一个参数，即要删除的应用程序角色的名称。例如：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DROP APPLICATION ROLE DW_App
```

解析

这个技巧演示了如何执行下列操作：

- 使用CREATE APPLICATION ROLE创建新的应用程序角色；
- 使用sp_setapprole激活角色权限；
- 使用ALTER APPLICATION ROLE修改应用程序角色；
- 使用DROP APPLICATION ROLE从数据库中删除应用程序角色。

应用程序角色对于那些希望授予仅通过应用程序的用户访问的应用程序开发者是便利的解决方案。聪明的最终用户经常会发现他们的SQL登录名也可以通过其他应用程序（比如Microsoft Access或SQL Server Management Studio）来连接到SQL Server。为了避免这个问题，可以改变登录账号以拥有数据库的最小权限，然后为必要的权限使用应用程序角色。这样的话，用户只能通过应用程序访问数据，而应用程序是使用应用程序角色来编程的。

499

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第18章 安全对象、权限和审核

在前一章中，我讨论了主体，它是可以访问SQL Server的安全账户。在这一章中，将讨论和演示安全对象（securable）和权限（permission）。安全对象是SQL Server控制访问权限的资源。SQL Server中的安全对象分为3种嵌套分级范围。层次结构的最高级是服务器范围。它包含登录名、数据库和端点。数据库范围包含在服务器范围中，控制着数据库用户、角色、安全凭证、架构等安全对象。第三个而且最里层的范围是架构范围，它控制安全对象（比如架构本身）及架构中的对象（比如表、视图、函数以及存储过程）。

权限允许主体在安全对象上执行操作。跨越所有的安全对象范围，用来控制主体到安全对象的访问的主要命令是GRANT、DENY和REVOKE。基于指定的安全对象范围，这些命令以相似的方式使用。GRANT用来启用对安全对象的访问。DENY显式地限制访问，禁止通常允许主体对安全对象访问的其他权限。REVOKE完全地删除安全对象上的指定权限，包括GRANT和DENY权限。

授予权限之后，可能仍然需要其他业务和授权跟踪变更或了解哪些登录名访问哪些表的符合性审核需求。为了解决这个需要，SQL Server 2008引入了SQL Server审核对象，可以使用它来收集SQL实例和感兴趣的数据库监视范围的操作。这个审核信息可以保存到文件、Windows应用程序事件日志或Windows安全事件日志中。

在本章中，将讨论如何将权限授予3种安全对象范围内的主体。除了权限以外，本章也会展示如下相关的安全对象和权限技巧：

- 如何使用CREATE、ALTER和DROP SCHEMA来管理架构；
- 如何通过使用函数fn_my_permissions报告指定主体上分配的权限；
- 如何使用系统函数Has_perms_by_name确定连接在安全对象上的权限，以及使用EXECUTE AS将连接的安全上下文定义到不同的登录名或用户来查看它们的权限；
- 如何使用sys.database_permissions和sys.server_permissions查询所有授予、拒绝和取消权限的情况；
- 如何使用ALTER AUTHORIZATION改变安全对象的拥有关系；
- 如何使用CREATE CREDENTIAL和ALTER LOGIN给SQL登录名提供Windows外部资源权限；
- 如何使用SQL Server 2008新的SQL Server审核功能来审核SQL实例——所有数据库级别的操作。

本章首先总体讨论SQL Server的权限。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

18.1　权限总览

这些权限在3个安全对象范围（服务器、数据库以及架构）中应用到SQL Server对象。SQL Server使用了一组应用到不同安全对象（以及不同的范围）的公用的权限名字，并且意味着对安全对象不同级别的授权。表18-1显示了可用于多个安全对象的一些权限（列表不完整）。

表18-1　重点权限

权　限	描　述
ALTER	启用被授予者对安全对象使用ALTER、CREATE或DROP命令的权限。例如，使用ALTER TABLE需要对指定的表的ALTER权限
AUTHENTICATE	启用在数据库或SQL Server范围中被信任的授予者
CONNECT	启用被授予者连接到SQL Server资源（比如端点或SQL Server实例）的权限
CONTROL	为被授予者启用对指定安全对象的所有可用权限，也包括其中所有嵌套或隐含的权限（例如，如果你CONTROL了一个架构，你也控制了这个架构内的所有表、视图或其他数据库对象）
CREATE	启用被授予者创建安全对象的权限（它可以是在服务器、数据库或架构范围内）
IMPERSONATE	启用被授予者模拟另一主体（登录名或用户）的权限。例如，对登录名使用EXECUTE AS命令时需要IMPERSONATE权限。我在10.2.2节的技巧中演示了使用EXECUTE AS。在本章中，我将继续讲解如何使用EXECUTE AS设置在模块之外的安全上下文
TAKE OWNERSHIP	允许被授权者获取所授予的安全对象的所有权
VIEW	启用被授予者查看关于指定安全对象的系统元数据的权限

使用系统目录表函数sys.fn_builtin_permissions可以报告SQL Server中的可用权限，也可以查看指定权限在权限层次结构中的位置。

语法如下：

```
  sys.fn_builtin_permissions
( [ DEFAULT | NULL ] | empty_string |
APPLICATION ROLE | ASSEMBLY | ASYMMETRIC KEY |
CERTIFICATE | CONTRACT | DATABASE |
ENDPOINT | FULLTEXT CATALOG| LOGIN |
MESSAGE TYPE | OBJECT | REMOTE SERVICE BINDING |
ROLE | ROUTE | SCHEMA | SERVER | SERVICE |
SYMMETRIC KEY | TYPE | USER | XML SCHEMA COLLECTION )
```

502

表18-2描述了这个命令的参数。

表18-2　fn_builtin_permissions的参数

权　限	描　述
DEFAULT \| NULL \| empty_string	在结果集中列出的所有权限中指定前面的这3个参数结果中的某一个
APPLICATION ROLE \| ASSEMBLY \| ASYMMETRIC KEY \| CERTIFICATE \|.CONTRACT \| DATABASE \| ENDPOINT \|FULLTEXT CATALOG\| LOGIN \| MESSAGE TYPE \| OBJECT \| REMOTE SERVICE BINDING \|ROLE \| ROUTE \| SCHEMA \| SERVER \| SERVICE \| SYMMETRIC KEY \| TYPE \| USER \| XML SCHEMA COLLECTION	为了返回某个类型的权限，指定其中某种安全对象类型

除权限名字之外，你可以通过查看covering_permission_name（在该类的权限，这个类是更常规权限的超集）、parent_class_desc（权限的父类——如果有的话）和结果集中的列parent_covering_permission_name（父覆盖权限——如果有的话）确定权限的嵌套层次结构，在下一技巧中将

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

会看到演示。

报告 SQL Server 可分配的权限

在这一技巧中，我会展示如何查看SQL Server中可用的权限，并且解释它们在权限层次结构中的位置。在第一个示例中，将返回所有的权限，而不管安全对象的范围：

```
SELECT class_desc, permission_name, covering_permission_name,
parent_class_desc, parent_covering_permission_name
FROM sys.fn_builtin_permissions(DEFAULT)
ORDER BY class_desc, permission_name
```

它返回如下的（部分）结果集：

class_desc	permission_name	covering_permission_name	parent_class_desc	parent_covering_permission_name
APPLICATION ROLE	ALTER	CONTROL	DATABASE	ALTER ANY APPLICATION ROLE
APPLICATION ROLE	CONTROL		DATABASE	CONTROL
APPLICATION ROLE	VIEW DEFINITION	CONTROL	DATABASE	VIEW DEFINITION
...				
SERVER	ALTER ANY DATABASE	CONTROL SERVER	...	...
...				
XML SCHEMA COLLECTION	REFERENCES	CONTROL	SCHEMA	REFERENCES
XML SCHEMA COLLECTION	TAKE OWNERSHIP	CONTROL	SCHEMA	CONTROL
XML SCHEMA COLLECTION	VIEW DEFINITION	CONTROL	SCHEMA	VIEW DEFINITION

下一个示例只显示了架构安全对象范围中的权限：

```
SELECT permission_name, covering_permission_name, parent_class_desc
FROM sys.fn_builtin_permissions('schema')
ORDER BY permission_name
```

它返回如下部分结果集：

permission_name	covering_permission_name	parent_class_desc
ALTER	CONTROL	DATABASE
CONTROL		DATABASE
DELETE	CONTROL	DATABASE
EXECUTE	CONTROL	DATABASE
INSERT	CONTROL	DATABASE
REFERENCES	CONTROL	DATABASE
SELECT	CONTROL	DATABASE
TAKE OWNERSHIP	CONTROL	DATABASE
UPDATE	CONTROL	DATABASE
VIEW CHANGE TRACKING	CONTROL	DATABASE
VIEW DEFINITION	CONTROL	DATABASE

解析

使用系统目录函数sys.fn_builtin_permissions可以查看SQL Server中的可用权限。

这个技巧中的第一个示例使用sys.fn_builtin_permissions和DEFAULT选项来显示所有权限。代码的第一行引用了从函数中返回的列名：

```
SELECT class_desc, permission_name, covering_permission_name,
parent_class_desc, parent_covering_permission_name
```

第二行引用了FROM子句中的函数，使用DEFAULT选项来显示所有权限：

```
FROM sys.fn_builtin_permissions(DEFAULT)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

代码最后一行对权限的类和名称进行排序：

```
ORDER BY class_desc, permission_name
```

结果显示了安全对象类的描述、权限名称以及覆盖权限名称（覆盖权限名称是在嵌套的权限层次结构中具有更高类的权限类的名称）。例如，对于APPLICATION ROLE类，可以看到CONTROL权限是DATABASE类和ALTER ANY APPLICATION权限的子类，但是不属于APPLICATION ROLE类中所有的覆盖权限（因为CONTROL为被授予者启用了指定的安全对象上的所有可用权限，也包括所有嵌套或隐式的权限）：

```
class_desc          permission_name  covering_permission_name  parent_class_desc  parent_covering_permission_name
...
APPLICATION ROLE    CONTROL          DATABASE                  CONTROL
...
```

对于OBJECT类，可以看到ALTER权限是SCHEMA父类和ALTER权限的子类。在OBJECT类中的ALTER权限
504 也是覆盖CONTROL权限的子类（正如在列convering_permission_name中所看到的）：

```
class_desc  permission_name  covering_permission_name  parent_class_desc  parent_covering_permission_name
...
OBJECT      ALTER            CONTROL                   SCHEMA             ALTER
...
```

对于SERVER类和ALTER ANY DATABASE权限，SERVER类的覆盖权限是CONTROL SERVER。注意SERVER类没有父类和权限：

```
class_desc  permission_name  covering_permission_name  parent_class_desc  parent_covering_permission_name
...
SERVER      ALTER ANY DATABASE                         CONTROL SERVER     ...  ...
...
```

技巧中的第二个示例只返回架构的安全对象类的权限。代码的第一行只包含3列：

```
SELECT permission_name, covering_permission_name, parent_class_desc
```

为了显示架构安全对象类的权限，第二行包含单词schema：

```
FROM sys.fn_builtin_permissions('schema')
```

然后以权限名称对结果排序：

```
ORDER BY permission_name
```

返回架构中控制数据库对象（比如视图、表，等等）的权限。例如，可以看到在架构范围中发现了DELETE权限，并且被CONTROL权限覆盖，它的父类是DATABASE安全对象：

```
permission_name     covering_permission_name    parent_class_desc
...
DELETE              CONTROL                     DATABASE
...
```

18.2　服务器范围的安全对象和权限

服务器范围的安全对象是唯一在SQL Server实例中的对象，包括端点、登录名和数据库。服务器范围安全对象的权限可以只授予服务器级别的主体（SQL Server登录名或Windows登录名），而不授予

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

数据库级别的主体，比如用户或数据库角色。

在权限层次结构的顶端，服务器权限允许被授予者执行一些活动，比如创建数据库、登录名或链接服务器。服务器权限允许关闭SQL Server实例（使用SHUTDOWN）或使用SQL Profiler（使用ALTER TRACE权限）。当将权限分配到安全对象的主体时，做这个分配操作的人称为授予者（grantor），接受权限的主体称为被授予者（grantee）。

授予服务器权限的部分语法如下：

```
GRANT Permission [ ,...n ]
TO  grantee_principal [ ,...n ]
[ WITH GRANT OPTION ]
[ AS grantor_principal ]
```

18
505

表18-3描述了这个命令的参数。

表18-3 GRANT的参数

参　数	描　述
Permission [,...n]	你可以在一个GRANT语句中授予一个或多个服务器权限
TO gratee_principal [,...n]	这里是被授予者，也是主体（SQL Server登录名），就是你要授予权限的人
WITH GRANT OPTION	当指定这个选项时，被授予者就拥有将权限授予其他被授予者的权限
AS grantor_principal	这个可选子句指定授予者派生它的权力，将权限授予被授予者

使用DENY命令可以显式地对服务器级别主体拒绝安全对象的权限。

语法如下：

```
DENY permission  [ ,...n ]
TO  grantee_principal [ ,...n ]
[ CASCADE ]
[ AS grantor_principal ]
```

表18-4描述了这个命令的参数。

表18-4 DENY的参数

参　数	描　述
permission [,...n]	要拒绝的一个或多个服务器范围权限
grantee_principal [,...n]	你可以拒绝权限到的一个或多个登录名（Windows或SQL）
CASCADE	当指定这个权限时，如果被授予者主体授予所有这些权限给其他主体，那么那些被授予者的权限也将被拒绝
AS grantor_principal	这个可选子句指定授予者派生它的权力，拒绝将权限给被授予者

使用REVOKE命令可以取消到主体的安全对象上的权限。取消权限的意思是你既没有授予也没有拒绝这个权限——取消操作删除以前授予或拒绝了的权限。

语法如下：

```
REVOKE [ GRANT OPTION FOR ]  permission   [ ,...n ]
FROM < grantee_principal > [ ,...n ]
 [ CASCADE ]
 [ AS grantor_principal ]
```

表18-5描述了这个命令的参数。

506

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表18-5　REVOKE的参数

参　数	描　述
GRANT OPTION FOR	当指定它时，被授予者将权限授予其他被授予者的权力将被取消
permission [,...n]	要取消的一个或多个服务器范围权限
grantee_principal [,...n]	要取消权限的一个或多个登录名（Windows或SQL）
CASCADE	当指定这个选项时，如果被授予者主体授予所有这些权限给其他主体，那些被授予者的权限也将被取消
AS grantor_principal	这个可选子句指定授予者派生它的权力，取消被授予者的权限

18.2.1　管理服务器权限

在这个技巧的第一个示例中，SQL登录名Veronica被授予使用SQL Profiler工具去监视SQL Server活动的能力。这个权限还给予了WITH GRANT OPTION，因此Veronica也可以GRANT权限给其他主体。要牢记在当前数据库为master时，可以只授予在服务器范围的权限，因此我们先进行切换数据库的上下文的批处理：

```
USE master
GO

-- 如果存在技巧登录名则创建它
IF NOT EXISTS
   (SELECT name
    FROM sys.server_principals
    WHERE name = 'Veronica')
BEGIN

    CREATE LOGIN [Veronica]
    WITH PASSWORD=N'test!#1',
    DEFAULT_DATABASE=[master],
    CHECK_EXPIRATION=OFF,
    CHECK_POLICY=OFF
END

GRANT ALTER TRACE TO Veronica
WITH GRANT OPTION
```

在第二个示例中，Windows登录名[JOEPROD\TestUser]被授予在SQL Server实例中创建和查看数据库的权限：

```
USE master
GO

GRANT CREATE ANY DATABASE, VIEW ANY DATABASE TO [JOEPROD\TestUser]
```

在下一个示例中，拒绝Windows登录名[JOEPROD\TestUser]执行SHUTDOWN命令的权力：

507

```
DENY SHUTDOWN TO [JOEPROD\TestUser]
```

在最后一个示例中，从Veronica取消使用SQL Profiler工具的权限，也包括所有她可能给予这个权限到的其他被授予者的权限：

```
USE master
GO
REVOKE ALTER TRACE FROM Veronica CASCADE
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

使用GRANT授予、使用DENY拒绝以及使用REVOKE取消服务器范围安全对象的权限。使用这些命令，在同一命令中可以指派一个或多个权限，也可以分配到一个或多个登录名上（Windows或SQL）。

本技巧讲解了在服务器范围分配权限，在之后的技巧中会看到分配数据库和架构权限的语法是非常相似的。

18.2.2 查询服务器权限

可以使用sys.server_permissions目录视图标识出SQL Server实例级别的权限。在这个技巧中，将查询所有与登录名为TestUser2相关联的权限。首先创建新登录名：

```
USE master
GO
CREATE LOGIN TestUser2
WITH PASSWORD = 'abcde1111111!'
```

然后，授予另外一个服务器范围的权限，以及拒绝服务器范围的权限：

```
DENY SHUTDOWN TO TestUser2

GRANT CREATE ANY DATABASE TO TestUser2
```

查询sys.server_permissions和sys.server_principals目录视图返回前面创建的新登录名的所有服务器范围权限：

```
SELECT p.class_desc,
       p.permission_name,
       p.state_desc
FROM sys.server_permissions p
INNER JOIN sys.server_principals s ON
    p.grantee_principal_id = s.principal_id
WHERE s.name = 'TestUser2'
```

这个查询返回：

```
class_desc      permission_name          state_desc
SERVER          CONNECT SQL              GRANT
SERVER          CREATE ANY DATABASE      GRANT
SERVER          SHUTDOWN                 DENY
```

即使只显式地执行了一个GRANT和一个DENY操作，由于创建了登录名，同时也隐式地授予了新登录名在SERVER范围的CONNECT权限。

508

解析

在这个技巧中，查询了sys.server_permissions和sys.server_principals目录视图来返回与新创建的登录名相关联的服务器范围权限。在SELECT子句中，返回了权限的类、权限名称和权限关联的状态：

```
SELECT p.class_desc,
       p.permission_name,
       p.state_desc
```

在FROM子句中，通过被授予者主体ID连接了两个目录视图。被授予者是授予或拒绝权限的目标接受者：

```
FROM sys.server_permissions p
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
INNER JOIN sys.server_principals s ON
    p.grantee_principal_id = s.principal_id
```

在WHERE子句中，指定了我希望查看权限的登录名的名称：

```
WHERE s.name = 'TestUser2'
```

18.3　数据库范围的安全对象和权限

数据库级别的安全对象对于指定的数据库是唯一的，并且包括几个SQL Server对象，比如角色、程序集、密码系统对象（密钥和证书）、Service Broker对象、全文目录、数据库用户、架构，等等。

可以给数据库主体（数据库用户、角色）授予这些安全对象的权限。授予数据库权限的部分语法如下所示：

```
GRANT permission [ ,...n ]
TO database_principal [ ,...n ]
[ WITH GRANT OPTION ]
[ AS database_principal ]
```

表18-6描述了这个命令的参数。

表18-6　GRANT的参数

参　数	描　述
Permission [,...n]	这个参数指定授予主体的一个或多个数据库权限
database_principal [,...n]	这个参数定义了被授予者新的权限
WITH GRANT OPTION	当指定了这个选项时，被授予者拥有将权限授予其他被授予者的权限
AS database_principal	这个可选子句指定授予者派生它的权力，将权限授予被授予者。例如，如果当前的数据库用户上下文没有GRANT指定权限的权限，但你在这个数据库用户上拥有IMPERSONATE权限，那么你就可以在AS子句中指定那个用户

509

使用DENY命令来拒绝将数据库范围的权限给被授予者。部分语法如下所示：

```
DENY permission [ ,...n ]
TO database_principal [ ,...n ] [ CASCADE ]
[ AS database_principal]
```

表18-7描述了这个命令的参数。

表18-7　DENY的参数

参　数	描　述
permission [,...n]	这个参数指定了要拒绝的一个或多个数据库范围的权限
<database_principal> [,...n]	这个参数定义了要拒绝授予一个或多个数据库主体权限
CASCADE	当指定这个选项时，如果被授予者主体授予了所有这些权限给其他主体，那么那些被授予者的权限也将被拒绝
AS database_principal	这个可选子句指定授予者派生它的权力，拒绝权限给被授予者

使用REVOKE命令来取消将数据库范围的权限给被授予者。部分语法如下所示：

```
REVOKE permission [ ,...n ]
FROM < database_principal > [ ,...n ]
[ CASCADE ]
[ AS database_principal]
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表18-8描述了这个命令的参数。

表18-8 REVOKE的参数

参 数	描 述
database_permission [,...n]	这个参数指定了要取消的一个或多个数据库范围的权限
<database_principal> [,...n]	这个参数定义了要取消给一个或多个数据库主体的权限
CASCADE	当指定这个选项时，如果被授予者主体授予了所有这些权限给其他主体，那么那些被授予者的权限也将被取消
AS database_principal	这个可选子句指定授予者派生它的权力，取消权限给被授予者

18.3.1 管理数据库权限

如果登录名和用户之前不存在或是在本章前面没有创建它，则在这个技巧中将首先创建登录名和用户：

```
-- 如果技巧的登录名不存在则创建它
IF NOT EXISTS (SELECT name FROM sys.server_principals WHERE name = 'Danny')
BEGIN
     CREATE LOGIN [Danny] WITH PASSWORD=N'test!#23',
     DEFAULT_DATABASE=[master], CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
END

-- 如果技巧的DB不存在则创建它
IF NOT EXISTS (SELECT name FROM sys.databases WHERE name = 'TestDB')
BEGIN

     CREATE DATABASE TestDB
END
GO

USE TestDB
GO

-- 如果数据库用户不存在则创建它们
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name = 'Veronica')
BEGIN
     CREATE USER Veronica FROM LOGIN Veronica
END

IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name = 'Danny')
BEGIN
     CREATE USER Danny FROM LOGIN Danny
END
```

这第一个示例演示了在数据库TestDB中将数据库权限授予数据库用户Veronica：

```
USE TestDB
GO
GRANT ALTER ANY ASSEMBLY, ALTER ANY CERTIFICATE
TO VERONICA
```

第二个示例演示了拒绝授予数据库用户Danny权限：

```
DENY ALTER ANY DATABASE DDL TRIGGER TO Danny
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

最后一个示例演示了取消用户Danny连接到数据库TestDB的数据库权限：

```
REVOKE CONNECT FROM Danny
```

解析

这个技巧演示了如何授予、取消或拒绝数据库范围的权限到数据库主体。你可能已经注意到，授予数据库范围的权限的语法几乎和授予服务器范围的权限一模一样。架构范围的权限也以相同的命令进行管理，但也有些细小的差别。

在介绍如何管理架构权限之前，在下一技巧中将演示在通常情况下如何管理架构。

18.3.2　查询数据库权限

可以使用sys.database_permissions目录视图来标识出数据库中的权限。在这个技巧中，将查询
511 所有与AdventureWorks数据库中名为TestUser的用户相关联的权限。我先创建新登录名和用户：

```
USE master
GO
CREATE LOGIN TestUser WITH PASSWORD = 'abcde1111111!'

USE AdventureWorks
GO
CREATE USER TestUser FROM LOGIN TestUser
```

下一步，授予和拒绝各种权限：

```
GRANT SELECT ON HumanResources.Department TO TestUser
DENY SELECT ON Production.ProductPhoto TO TestUser
GRANT EXEC ON HumanResources.uspUpdateEmployeeHireInfo TO TestUser
GRANT CREATE ASSEMBLY TO TestUser
GRANT SELECT ON Schema::Person TO TestUser
DENY IMPERSONATE ON USER::dbo TO TestUser
DENY SELECT ON HumanResources.Employee(Birthdate) TO TestUser
```

现在查询sys.database_principals目录视图来确定主体标识符：

```
SELECT principal_id
FROM sys.database_principals
WHERE name = 'TestUser'
```

这个查询返回如下结果（如果你尝试这个技巧，要记住你的主体标识符可能会不同）：

```
principal_id
5
```

现在我可以使用主体ID为5的作为sys.database_permissions表的被授予者主体ID（尽管可以将前一个查询整合到下一个查询中，但为了清晰呈现目录视图的作用，这里将其分别给出）：

```
SELECT
   p.class_desc,
   p.permission_name,
   p.state_desc,
   ISNULL(o.type_desc,'') type_desc,
   CASE p.class_desc
      WHEN 'SCHEMA'
         THEN schema_name(major_id)
      WHEN 'OBJECT_OR_COLUMN'
         THEN CASE
                 WHEN minor_id = 0
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                THEN object_name(major_id)
            ELSE (SELECT
                    object_name(object_id) +
                    '.'+
                    name
                FROM sys.columns
                WHERE object_id = p.major_id AND
                      column_id = p.minor_id) END
        ELSE '' END AS object_name
FROM sys.database_permissions p
LEFT OUTER JOIN sys.objects o ON
    o.object_id = p.major_id
WHERE grantee_principal_id = 5
```

这个查询返回：

```
class_desc          permission_name    state_desc   type_desc              object_name
DATABASE            CONNECT            GRANT
DATABASE            CREATE ASSEMBLY    GRANT
OBJECT_OR_COLUMN    SELECT             GRANT        USER_TABLE             Department
OBJECT_OR_COLUMN    SELECT             DENY         USER_TABLE             Employee.BirthDate
OBJECT_OR_COLUMN    EXECUTE            GRANT        SQL_STORED_PROCEDURE   uspUpdateEmployee
                                                                           HireInfo
OBJECT_OR_COLUMN    SELECT             DENY         USER_TABLE             ProductPhoto
SCHEMA              SELECT             GRANT                               Person
DATABASE_PRINCIPAL  IMPERSONATE        DENY
(8行受影响)
```

解析

这个技巧演示了查询系统目录视图来确定指定数据库用户的权限。我创建登录名和用户，然后授予和拒绝各种权限给它。

在这之后，我查询sys.database_principals目录视图来确定新用户的ID。

一点点阅读技巧中最后一个稍微复杂的查询，查询前面几列显示了类描述、权限名称以及状态（例如，GRANT或DENY）：

```
SELECT
   p.class_desc,
   p.permission_name,
   p.state_desc,
```

类型描述实际是从sys.objects视图中获得的，用它来获取关于对象的权限信息。如果它为NULL，则不在结果集中返回字符：

```
ISNULL(o.type_desc,'') type_desc,
```

下一个表达式是运算类描述的CASE语句。当类为架构时，返回架构的名称：

```
CASE p.class_desc
   WHEN 'SCHEMA'
      THEN schema_name(major_id)
```

当类为对象或列时，嵌套另一个CASE语句：

```
WHEN 'OBJECT_OR_COLUMN'
   THEN CASE
```

如果minor_id为0，就会知道它是一个对象而不是列，因此返回对象名：

```
WHEN minor_id = 0
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    THEN object_name(major_id)
```

否则，处理列名，因此执行子查询将对象名和列名拼接起来：

```
ELSE (SELECT
        object_name(object_id) +
        '.'+
      name
    FROM sys.columns
    WHERE object_id = p.major_id AND
          column_id = p.minor_id) END
ELSE '' END AS object_name
```

513

使用LEFT OUTER JOIN命令在sys.object视图上查询权限。因为不是所有权限都与对象关联，所以没有使用INNER联结。例如，在CREATE ASSEMBLY权限上的GRANT：

```
FROM sys.database_permissions p
LEFT OUTER JOIN sys.objects o ON
  o.object_id = p.major_id
```

最后，限定被授予者是创建的用户的ID。被授予者是权限的接受者。sys.database_permissions也有grantor_principal_id，它是指定行的权限授予者。我不希望指定它——我只是想将权限授予指定用户的行：

```
WHERE grantee_principal_id = 5
```

18.4 架构范围的安全对象和权限

架构范围的安全对象包含在数据库安全对象范围内，并且包括用户定义数据类型、XML架构集合以及对象。对象安全对象也在它之内拥有其他类型的安全对象，本章稍后会介绍。

在SQL Server 2005和2008中，用户从直接数据库对象的所有权（比如表、视图和存储过程）中分离出来。通过使用架构实现了这种分离，从根本上说，它是数据库对象的容器。不是直接的对象拥有者，对象包含在架构中，并且这个架构被用户所拥有。

一个或多个用户可以拥有架构，或使用它当作他们创建对象的默认架构。而且，可以在架构级别应用安全措施。这意味着架构里的所有对象可以作为一个整体进行管理，而不是以单个对象级别进行管理。

每个数据库都有dbo架构，如果没有指定默认架构，对象将会创建在这里。但如果希望创建你自己的架构，可以使用CREATE SCHEMA命令。

部分语法如下：

```
CREATE SCHEMA schema_name [AUTHORIZATION owner_name ]
```

表18-9描述了这个命令的参数。

表18-9 CREATE SCHEMA的参数

参 数	描 述
schema_name	架构名
owner_name	拥有者是数据库主体，它可以在数据库中拥有一个或多个架构

使用DROP SCHEMA命令可以删除既有的架构。

语法如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

514

```
DROP SCHEMA schema_name
```

这个命令只有一个参数：要从数据库中删除的架构名。而且，不可以删除包含对象的架构，因此必须删除对象或将其转移到新架构中。

注解 要回顾使用ALTER SCHEMA命令转移对象的架构所有权的方法，请看24.1.2节。

像服务器范围和数据库范围的权限一样，架构的权限也使用GRANT、DENY、REVOKE命令来管理。在架构上授予权限的部分语法如下：

```
GRANT permission [ ,...n ] ON SCHEMA :: schema_name
TO database_principal [ ,...n]
[ WITH GRANT OPTION ][ AS granting_principal ]
```

表18-10描述了这个命令的参数。

表18-10 GRANT的参数

参 数	描 述
permission [,...n]	这个参数指定要授予被授予者的一个或多个架构权限
schema_name	这个参数定义被授予者接受的权限架构名
database_principal	这个参数指定了数据库主体权限接受者
WITH GRANT OPTION	当指定这个选项时，被授予者则拥有将架构权限授予其他被授予者的权限
AS granting_principal	这个可选子句指定授予者派生它的权力，将架构范围的权限授予被授予者

使用DENY命令可以拒绝将架构范围的权限给被授予者。

部分语法如下：

```
DENY permission [ ,...n ] ON SCHEMA :: schema_name
TO  database_principal [ ,...n ]
 [ CASCADE ]
 [ AS denying_principal ]
```

表18-11描述了这个命令的参数。

表18-11 DENY的参数

参 数	描 述
permission [,...n]	这个参数指定要拒绝的一个或多个架构范围的权限
schema_name	这个参数定义要拒绝权限的架构名
database_principal [,...n]	这个参数指定要拒绝其权限的一个或多个数据库主体
CASCADE	当指定这个选项时，如果被授予者主体授予了所有这些权限给其他主体，那些被授予者的权限也将被拒绝
AS denying_principal	这个可选子句指定授予者派生它的权力，拒绝将权限给被授予者

515

使用REVOKE命令可以取消架构范围的权限。部分语法如下：

```
REVOKE [ GRANT OPTION FOR ]
permission [ ,...n ]
 ON SCHEMA :: schema_name
{ TO | FROM } database_principal [ ,...n ]
 [ CASCADE ] [ AS principal ]
```

表18-12描述了这个命令的参数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表18-12　REVOKE的参数

参　数	描　述
permission [,...n]	这个参数指定了要取消的一个或多个架构范围的权限
schema_name	这个参数定义了要取消权限的架构名
database_principal [,...n]	这个参数定义要取消其权限的一个或多个数据库主体
CASCADE	当指定这个选项时，如果被授予者主体授予了所有这些权限给其他主体，那么那些被授予者的权限也将被取消
AS principal	这个可选子句指定授予者派生它的权力，取消将权限给被授予者

18.4.1　管理架构

在这一技巧中，在数据库TestDB中创建一个新的、名为Publishers的架构：

```
USE TestDB
GO
CREATE SCHEMA Publishers AUTHORIZATION db_owner
```

现在我拥有一个名为Publishers的架构，它可用来容纳其他数据库对象。例如，它可以用来保存所有与发布功能相关的对象，或用来保存关联到发布活动的数据库用户。

从使用新架构开始，使用两部分组成名称的命名格式的架构schema.object_name：

```
CREATE TABLE Publishers.ISBN
(ISBN char(13) NOT NULL PRIMARY KEY,
 CreateDT datetime NOT NULL DEFAULT GETDATE())
GO
```

516

下一个示例演示了使架构Publishers为一个数据库用户的默认架构。在这个示例中，将在数据库master中创建新SQL登录名：

```
USE master
GO
CREATE LOGIN Nancy
WITH PASSWORD=N'test123',
     DEFAULT_DATABASE=TestDB,
     CHECK_EXPIRATION=OFF,
     CHECK_POLICY=OFF
GO
```

下一步，在数据库TestDB中创建新数据库用户：

```
USE TestDB
GO
CREATE USER Nancy FOR LOGIN Nancy
GO
```

现在，既有数据库用户的默认架构将改变为架构Publishers。这个数据库用户默认创建的所有对象都将属于这个架构（除非数据库用户在对象创建的语句中显式使用了不同的架构）：

```
USE TestDB
GO
ALTER USER Nancy WITH DEFAULT_SCHEMA=Publishers
GO
```

第24章介绍了如何使用ALTER SCHEMA将对象的所有权从一个架构转移到另一个架构。在希望删除架构的情况下，需要使用这个操作。例如，如果马上尝试删除架构Publishers，但在架构中保留表

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Publishers.ISBN，将会有错误警告，错误为有对象引用了该架构。这个示例演示了在从数据库中删除架构Publishers之前，使用ALTER SCHEMA将表转移到dbo架构中。

```
ALTER SCHEMA dbo TRANSFER Publishers.ISBN
GO

DROP SCHEMA Publishers
```

解析

架构是数据库对象的容器。不像数据库用户直接拥有对象，现在数据库用户可以拥有架构（或换句话来说，拥有使用架构中对象的权限）。

在这个技巧中，使用CREATE SCHEMA创建了一个名为Publishers的新架构。在这个新架构中创建了一个名为Publishers.ISBN的新表。在这之后，在数据库TestDB中创建了新的登录名和数据库用户。通过ALTER USER使新架构为新用户的默认架构。

在架构中的所有对象被删除或是转换到其他地方之前，不可以删除这个架构，使用ALTER SCHEMA将Publishers.ISBN表转移到dbo架构中。然后使用DROP SCHEMA从数据库中删除架构Publishers。

18.4.2 管理架构的权限

在下面一组示例中，将展示如何管理架构权限。在展示之前，先快速讲一下如何知道数据库中存在哪些架构。可以查询系统目录视图sys.schemas来查看数据库中的架构。这个示例演示了列出数据库AdventureWorks中存在的架构：

517

```
USE AdventureWorks
GO

SELECT s.name SchemaName, d.name SchemaOwnerName
FROM sys.schemas s
INNER JOIN sys.database_principals d ON
   s.principal_id= d.principal_id
ORDER BY s.name
```

它返回内建数据库架构的列表（固定数据库角色、dbo、guest、sys以及INFORMATION_SCHEMA）以及用户定义的架构（Person、Production、Purchasing、Sales、HumanResources）：

```
SchemaName              SchemaOwnerName
db_accessadmin          db_accessadmin
db_backupoperator       db_backupoperator
db_datareader           db_datareader
db_datawriter           db_datawriter
db_ddladmin             db_ddladmin
db_denydatareader       db_denydatareader
db_denydatawriter       db_denydatawriter
db_owner                db_owner
db_securityadmin        db_securityadmin
dbo                     dbo
guest                   guest
HumanResources              dbo
INFORMATION_SCHEMA      INFORMATION_SCHEMA
Person                  dbo
Production              dbo
Purchasing              dbo
Sales                   dbo
sys                     sys
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在数据库AdventureWorks中，现在演示在架构上分配权限给数据库主体。在这个示例中，数据库用户TestUser被授予TAKE OWNERSHIP权限到架构Person，它使被授予者拥有了被授予的安全对象的所有权：

```
GRANT TAKE OWNERSHIP
ON SCHEMA ::Person
TO TestUser
```

在下一个示例中，在同一个语句中授予数据库用户TestUser多个权限，包括ALTER架构、在架构中EXECUTE存储过程或从架构中的表或视图进行SELECT的能力。使用WITH GRANT OPTION后，TestUser也可以将这些权限授予其他数据库主体：

```
GRANT ALTER, EXECUTE, SELECT
ON SCHEMA  ::Production
TO TestUser
WITH GRANT OPTION
```

在下一个示例中，拒绝数据库用户TestUser从架构Production的任何表中INSERT、UPDATE或DELETE
518 数据：

```
DENY INSERT, UPDATE, DELETE
ON SCHEMA ::Production
TO TestUser
```

在这个技巧的最后一个示例中，取消了TestUser的ALTER架构Production或从架构Production中的对象SELECT的权利，也取消了她可能授予其他人的这些权限（使用CASCADE）：

```
REVOKE ALTER, SELECT
ON SCHEMA ::Production
TO TestUser
CASCADE
```

解析

授予、拒绝或取消权限与用在数据库级别和服务器级别范围的权限使用的命令相同。但是有一点不同之处，它引用了ON SCHEMA，用它来指定授予、拒绝、取消权限的目标的架构名。还需要注意的是，架构名使用了两个冒号作为前缀（称为范围限定符）。范围限定符用来限定指定对象类型的权限范围。

18.5　对象的权限

对象是嵌套在架构范围中的，它们包括表、视图、存储过程、函数和聚合。在架构范围（比如SELECT或EXECUTE）定义权限可以给被授予者定义架构中所有对象的权限。也可以在对象级别定义权限。对象权限是在架构权限中嵌套的，数据库范围的权限中的架构权限以及服务器级别权限中的数据库范围权限。

像服务器级别、数据库范围以及架构范围的权限，可以使用GRANT、DENY以及REVOKE来定义指定数据库对象的权限。

部分授予对象权限的语法如下：

```
GRANT permission ON
[ OBJECT :: ][ schema_name ]. object_name [ ( column [ ,...n ] ) ]
TO <database_principal> [ ,...n ]
[ WITH GRANT OPTION ] [ AS database_principal ]
```

表18-13描述了这个命令的参数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表18-13 GRANT的参数

参 数	描 述
permission [,...n]	这个参数指定了要授予被授予者的一个或多个对象权限
[OBJECT ::][schema_name]. object_name [(column[,...n])]	这个参数指定了要授予权限的目标对象（或者为适当的列）
database_principal	这个参数指定接受权限的数据库主体
WITH GRANT OPTION	当指定这个选项时，被授予者拥有将权限授予其他被授予者的权限
AS database_principal	这个可选子句指定授予者派生它的权力，将架构范围的权限给被授予者

519

使用DENY命令可以拒绝对象权限到被授予者。部分语法如下：

```
DENY permission [ ,...n ] ON
 [ OBJECT :: ][ schema_name ]. object_name [ ( column [ ,...n ] ) ]
TO <database_principal> [ ,...n ]
[ CASCADE ] [ AS <database_principal> ]
```

表18-14描述了这个命令的参数。

表18-14 DENY的参数

参 数	描 述
[OBJECT ::][schema_name]. object_name [(column [,...n])]	这个参数指定拒绝权限的目标对象（或者为适当的列）
<database_principal> [,...n]	这个参数指定了要拒绝其权限的一个或多个数据库主体
CASCADE	当指定这个选项时，如果被授予者主体授予了所有这些权限给其他主体，那么被授予者的权限也将被拒绝
As database_principal	这个可选子句指定授予者派生它的权力，拒绝将权限给被授予者

使用REVOKE命令可以为被授予者取消对象权限。

部分语法如下：

```
REVOKE [ GRANT OPTION FOR ]
permission [ ,...n ]
ON [ OBJECT :: ][ schema_name ]. object_name [ ( column [ ,...n ] ) ]
FROM <database_principal> [ ,...n ]
 [ CASCADE ] [ AS <database_principal> ]
```

表18-15描述了这个命令的参数。

520

表18-15 REVOKE的参数

参 数	描 述
GRANT OPTION FOR	当使用这个选项时，取消授予权限到其他数据库主体的权利
permission [,...n]	这个参数指定要对被授予者取消的一个或多个对象的权限
[OBJECT ::][schema_name]. object_name [(column [,...n])]	这个参数指定取消权限的目标对象（或者为适当的列）
<database_principal> [,...n]	这个参数指定要取消其权限的一个或多个数据库主体
CASCADE	当指定这个选项时，如果被授予者主体授予了所有这些权限给其他主体，那么那些被授予者的权限也将被取消
AS database_principal	这个可选子句指定授予者派生它的权力，取消将权限给被授予者

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

管理对象的权限

在这个技巧中，授予数据库用户TestUser在表HumanResources.Department中SELECT、INSERT、DELETE和UPDATE数据的权限。

```
USE AdventureWorks
GO
GRANT DELETE, INSERT, SELECT, UPDATE
ON HumanResources.Department
TO TestUser
```

在这里，名为ReportViewers的数据库角色被授予了执行过程的能力，以及在系统目录视图中查看关于指定对象元数据的能力：

```
CREATE ROLE ReportViewers

GRANT EXECUTE, VIEW DEFINITION
ON dbo.uspGetManagerEmployees
TO ReportViewers
```

在下一个示例中，拒绝数据库用户TestUser对表HumanResources.Department的ALTER权限：

```
DENY ALTER ON HumanResources.Department TO TestUser
```

在最后一个示例中，取消TestUser对表HumanResources.Department的INSERT、UPDATE和DELETE的权限：

```
REVOKE INSERT, UPDATE, DELETE
ON HumanResources.Department
TO TestUser
```

521

解析

这个技巧演示了授予对象权限到指定数据库安全对象的操作。通过指定特定对象名和应用到这个对象的权限来授予对象权限。例如，EXECUTE权限可以授予存储过程，但SELECT权限不能。

权限可能被其他类型的权限取代。例如，如果数据库用户TestUser被授予了在表HumanResources.Department上的SELECT权限，但拒绝了在架构HumanResources本身的权限，因为DENY覆盖了所有GRANT SELECT的权限，当她尝试去SELECT这个表时她将收到如下的错误信息：

```
消息229，级别14，状态5，第2行
拒绝了对对象 'Department'（数据库'AdventureWorks'，架构'HumanResources'）的 SELECT 权限。
```

18.6　管理权限访问安全对象范围

现在已经回顾了各种安全对象范围以及将权限授予主体的方法，在下一组技巧中将展示如何报告和管理在不同范围的安全对象上主体拥有的权限。

18.6.1　检测当前连接的安全对象的权限

由于SQL Server的安全对象权限的嵌套层次结构（服务器、数据库和架构），权限可以从更高的范围继承下来。要确定当前的登录名/数据库连接拥有安全对象的何种权限是困难的，尤其是当添加服务器或数据库角色到等式中时。

要理解通过使用函数Has_perms_by_name可以检测增加到安全对象上的数据库连接拥有何种权限。如果当前用户被授予了权限，这个系统标量函数将返回1，否则返回0。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个函数的语法如下：

```
Has_perms_by_name ( securable , securable_class , permission
      [ , sub-securable ] [ , sub-securable_class ] )
```

表18-16描述了这个函数的参数。

表18-16 Has_perms_by_name的参数

参 数	描 述
securable	希望验证权限的安全对象的名称
securable_class	要检查的安全对象类的名称。类名（例如DATABASE或SCHEMA）可以从函数sys.fn_builtin_permissions的列class_desc中返回（请看本章的开始部分）
permission	要检查的权限的名称
sub-securable	安全对象子实体的名称
sub-securable_class	安全对象子实体类的名称

522

这个示例演示了如何检查当前连接的用户是否拥有对数据库AdventureWorks的ALTER权限：

```
USE AdventureWorks
GO
SELECT Has_perms_by_name ('AdventureWorks', 'DATABASE', 'ALTER')
```

它返回0，意思是当前连接没有对数据库AdventureWorks的ALTER权限：

```
0
```

下一个查询测试了当前连接，以查看是否可以对表Person.Address进行更新或查询：

```
SELECT  CASE Has_perms_by_name ('Person.Address', 'OBJECT', 'UPDATE')
          WHEN 1 THEN 'Yes'
          ELSE 'No'
        END UpdateTable,
        CASE Has_perms_by_name ('Person.Address', 'OBJECT', 'SELECT')
          WHEN 1 THEN 'Yes'
          ELSE 'No'
        END SelectFromTable
```

这个查询返回：

```
UpdateTable          SelectFromTable
Yes                  No
```

解析

系统函数Has_perms_by_name计算当前连接是否被授予访问指定安全对象的权限（授予权限可以是显式指定的，或从更高范围安全对象中继承下来的）。在这个技巧的两个示例中，使用的第一个参数为安全对象的名称（数据库名称或表名称）。第二个参数是安全对象类，如OBJECT或DATABASE。使用的第三个参数是实际要验证的权限，例如ALTER、UPDATE或SELECT（取决于被检查的安全对象的那些可用的权限）。

18.6.2 通过安全对象范围报告主体的权限

在这个技巧中，将演示使用函数fn_my_permissions返回当前连接的主体分配的权限。这个函数的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

语法如下：

```
fn_my_permissions ( securable , 'securable_class')
```

523 表18-17描述了这个命令的参数。

表18-17　fn_my_permissions的参数

参　数	描　述
securable	要验证的安全对象的名称。如果在服务器或数据库范围检查权限就使用NULL
securable_class	你要对其列出权限的安全对象类

第一个示例演示了检查当前连接的服务器范围权限：

```
SELECT permission_name
FROM fn_my_permissions(NULL, N'SERVER')
ORDER BY permission_name
```

它返回如下的结果（这个查询示例在sysadmin的上下文中执行，因此在这个例子中会返回所有可用的服务器范围的权限）：

```
ADMINISTER BULK OPERATIONS
ALTER ANY CONNECTION
ALTER ANY CREDENTIAL
ALTER ANY DATABASE
ALTER ANY ENDPOINT
ALTER ANY EVENT NOTIFICATION.
ALTER ANY LINKED SERVER
ALTER ANY LOGIN
ALTER RESOURCES
ALTER SERVER STATE
ALTER SETTINGS
ALTER TRACE
AUTHENTICATE SERVER
CONNECT SQL
CONTROL SERVER
CREATE ANY DATABASE
CREATE DDL EVENT NOTIFICATION
CREATE ENDPOINT
CREATE TRACE EVENT NOTIFICATION
EXTERNAL ACCESS ASSEMBLY
SHUTDOWN
UNSAFE ASSEMBLY
VIEW ANY DATABASE
VIEW ANY DEFINITION
VIEW SERVER STATE
```

如果登录名或数据库用户拥有IMPERSONATE权限，除了检查自己拥有的权限，也可以通过使用EXECUTE AS命令检查其他主体的权限。在第10章中演示了如何使用EXECUTE AS去指定存储过程的安全上下文。也可以以独立的方式使用EXECUTE AS。使用它来切换当前数据库会话的安全上下文。然后你可以通过运行REVERT命令切换回最初的安全上下文。

部分EXECUTE AS的语法如下：

```
EXECUTE AS { LOGIN | USER } = 'name'
[ WITH { NO REVERT } ]
```

524 表18-18描述了这个命令的参数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表18-18 EXECUTE AS部分语法参数

参 数	描 述
{ LOGIN \| USER } = 'name'	选择LOGIN模拟SQL或Windows登录名或USER模拟数据库用户。使用实际的登录名或用户名称
NO REVERT	如果指定了NO REVERT，就不可以使用REVERT命令切换回最初的安全上下文

这次通过使用登录名Veronica的安全上下文重新执行之前的查询来演示EXECUTE AS：

```
USE master
GO

EXECUTE AS LOGIN = N'Veronica'
GO

SELECT permission_name
FROM fn_my_permissions(NULL, N'SERVER')
ORDER BY permission_name
GO

REVERT
GO
```

由于没有在拥有sysadmin权限的登录名下调用，它返回了非常少的服务器权限列表：

```
CONNECT SQL
VIEW ANY DATABASE
```

下一个示例演示了返回数据库用户Veronica的数据库范围的权限：

```
USE TestDB
GO

EXECUTE AS USER = N'Veronica'
GO

SELECT permission_name
FROM fn_my_permissions(N'TestDB', N'DATABASE')
ORDER BY permission_name
GO

REVERT
GO
```

这个查询返回：

```
ALTER ANY ASSEMBLY
ALTER ANY CERTIFICATE
CONNECT
CREATE ASSEMBLY
CREATE CERTIFICATE
```

在下一个示例中，检查当前连接在表Production.Culture上的权限。这次展示表的所有子实体（意思是在表列上所有显式的权限）：

```
USE AdventureWorks
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SELECT subentity_name, permission_name
FROM fn_my_permissions(N'Production.Culture', N'OBJECT')
ORDER BY permission_name, subentity_name
```

它返回如下的结果（当填充了subentity_name时，这是列引用）：

```
subentity_name      permission_name
                    ALTER
                    CONTROL
                    DELETE
                    EXECUTE
                    INSERT
                    RECEIVE
                    REFERENCES
CultureID           REFERENCES
ModifiedDate        REFERENCES
Name                REFERENCES
                    SELECT
CultureID           SELECT
ModifiedDate        SELECT
Name                SELECT
                    TAKE OWNERSHIP
                    UPDATE
CultureID           UPDATE
ModifiedDate        UPDATE
Name                UPDATE
                    VIEW CHANGE TRACKING
                    VIEW DEFINITION
```

解析

这个技巧演示了如何使用函数fn_my_permissions返回当前连接的权限。第一个示例在第一个参数中使用了NULL，在第二个参数中使用了SERVER来返回当前连接的服务器范围的权限：

```
...
FROM fn_my_permissions(NULL, N'SERVER')
```

然后使用EXECUTE AS来执行相同的查询，这一次是在登录名Veronica的上下文中执行的，它为她的登录名返回服务器范围的权限：

```
EXECUTE AS LOGIN = N'Veronica'
GO
...

REVERT
GO
```

526

下一个示例展示了通过在第一个参数中指定数据库名称，在第二个参数中指定DATABASE来显示数据库范围的权限：

```
FROM fn_my_permissions(N'TestDB', N'DATABASE')
```

最后一个示例检查了当前连接在指定表上的权限：

```
...
FROM fn_my_permissions(N'Production.Culture', N'OBJECT')
```

它在表级别以及列级别返回信息。例如，ALTER和CONTROL权限在表级别是可用的，填充了entity_name（例如CultureID和ModifiedDate）的那些行引用表的列级别权限。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

18.6.3 改变安全对象所有权

正如本章之前讨论的，对象包含在架构中，架构被数据库用户或角色所拥有。改变架构的拥有者不再意味着必须对这个架构中的对象重命名。先不管架构，在SQL Server实例中的其他安全对象仍然通过服务器或数据库级别的主体拥有直接所有权。

例如，架构具有数据库主体的拥有者（比如数据库用户），端点具有服务器级别的拥有者，比如SQL登录名。

假定登录名在适当的TAKE OWNERSHIP权限下执行操作，可以使用ALTER AUTHORIZATION命令来改变安全对象的所有权。

部分ALTER AUTHORIZATION的语法如下：

```
ALTER AUTHORIZATION
ON [ <entity_type> :: ] entity_name
TO { SCHEMA OWNER | principal_name }
```

表18-19描述了这个命令的参数。

表18-19 ALTER AUTHORIZATION的参数

参 数	描 述
entity_type	这个参数指定了要给予新拥有者的安全对象的类
entity_name	这个参数指定了安全对象的名称
SCHEMA OWNER \| principal_name	这个参数表示新架构拥有者的名称，或具有安全对象所有权的数据库或服务器主体的名称

在这个示例中，将架构HumanResources的拥有者改变为数据库用户TestUser：

```
USE AdventureWorks
GO

ALTER AUTHORIZATION ON Schema::HumanResources
TO TestUser
```

在第二个示例中，端点的拥有者改变为SQL登录名。在进行此操作之前，使用系统目录视图sys.endpoints和sys.server_principals来检验既有的端点拥有者：

527

```
SELECT p.name OwnerName
FROM sys.endpoints e
INNER JOIN sys.server_principals p ON
   e.principal_id = p.principal_id
WHERE e.name = 'ProductWebsite'
```

它返回：

```
OwnerName
JOEPROD\Owner
```

下一步，将拥有者改变为不同的SQL登录名：

```
ALTER AUTHORIZATION ON Endpoint::ProductWebSite TO TestUser
```

再一次执行基于sys.server_principals和sys.endpoints的查询，显示新的拥有者：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
OwnerName
TestUser
```

解析

这个技巧演示了如何改变对象所有权。当需要删除登录名或数据库用户时，可能希望改变所有权。如果这个登录名或数据库用户拥有安全对象，可以在删除SQL登录名或数据库用户之前使用ALTER AUTHORIZATION来改变安全对象的拥有者。

在这个技巧中，使用ALTER AUTHORIZATION来将架构的拥有者改为不同的数据库用户，以及将端点的拥有者改为不同的SQL登录名（关联到Windows账户）。在这两个例子中，安全对象的名称加了::范围限定符前缀，它指定了改变所有权的对象类型。

18.6.4　允许SQL登录名访问非SQL Server的资源

在本章中，已经讨论了SQL Server实例中的权限和安全对象，尽管有时SQL登录名（没有关联到Windows用户或用户组）可能需要SQL Server实例外部的权限。Windows主体（Windows用户或用户组）拥有隐含的SQL Server实例外部的权限，但是SQL登录名没有，因为SQL登录名和密码是在SQL Server内部创建的。为了解决这个问题，可以将SQL登录名绑定到Windows凭据，将凭据的隐含的Windows权限给予SQL登录名。然后这个SQL登录名可以使用更高级的SQL Server功能，包括可能需要访问外部资源。这个凭据可以绑定到多个SQL登录名上（尽管一个SQL登录名可以绑定到单个凭据上）。

使用CREATE CREDENTIAL命令来创建凭据。

语法如下：

```
CREATE CREDENTIAL credential_name WITH IDENTITY = ' identity_name '
    [ , SECRET = ' secret ' ]
[ FOR CRYPTOGRAPHIC_PROVIDER cryptographic_provider_name ]
```

528　表18-20描述了这个命令的参数。

表18-20　CREATE CREDENTIAL的参数

参　数	描　述
credential_name	新凭据的名称
identity_name	外部账户名称（例如，Windows用户）
secret	凭据的密码
cryptographic_provider_name	企业密钥管理（EKM）提供程序的名称（当凭据有对应EKM提供程序时使用）

在这个示例中，创建一个映射到Windows用户账户JOEPROD\Owner的新凭据：

```
USE master
GO

CREATE CREDENTIAL AccountingGroup
WITH IDENTITY = N'JOEPROD\AccountUser1',
SECRET = N'mypassword!
```

在创建之后，在CREATE LOGIN和ALTER LOGIN中使用关键字CREDENTIAL，可以将凭据绑定到既有的或新的SQL登录名上：

```
USE master
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ALTER LOGIN Veronica
WITH CREDENTIAL = AccountingGroup
GO
```

解析

凭据允许SQL身份验证登录名绑定到Windows外部的权限。在这个技巧中，创建了名为AccountingGroup的新凭据。它映射到Windows用户JOEPROD\AccountUser1，并且在命令的SECRET参数中给出密码。在创建了凭据之后，通过使用ALTER LOGIN和WITH CREDENTIAL将其绑定到SQL登录名Veronica。使用凭据，现在登录名Veronica拥有与Windows账户 JOEPROD\AccountUser1相同的SQL Server外部的权限。

18.7 对安全对象的主体审核 SQL 实例和数据库级别活动

SQL Server 2008企业版引入了审核SQL实例和数据库级别活动的原生能力。使用服务器审核对象将这个活动捕获到目标数据目的地中，这个对象定义将活动捕获到文件、Windows应用程序事件日志、或是Windows安全事件日志中。服务器审核对象也允许指定如果不能写入到目标，是否关闭SQL Server实例。

创建服务器审核对象之后，可以将服务器审核规范或数据库审核规范对象绑定到服务器审核对象上。使用服务器审核规范来定义在SQL Server实例范围中希望捕获哪些事件。使用数据库审核规范对象可以在数据库范围内定义希望捕获哪些事件。只允许一个服务器审核规范绑定到一个服务器审核对象上，而一个或更多数据库审核规范可以绑定到一个服务器审核对象上。一个服务器审核对象可以同时配置一个服务器审核规范以及一个或更多数据库审核规范。

529

在下面的几个技巧中，将演示如何创建将捕获事件数据写入目标文件的服务器审核对象。然后演示如何将SQL实例级别和数据库级别事件关联到审核文件中，还会演示如何读取包含在二进制文件中的审核数据。

18.7.1 定义审核数据源

配置SQL Server 2008企业版审核的第一步是创建服务器审核对象。使用CREATE SERVER AUDIT命令来完成它。这个命令的语法如下：

```
CREATE SERVER AUDIT audit_name
    TO { [ FILE (<file_options> [, ...n]) ] | APPLICATION_LOG | SECURITY_LOG }
    [ WITH ( <audit_options> [, ...n] ) ]
}
[ ; ]
<file_options>::=
{
        FILEPATH = 'os_file_path'
    [, MAXSIZE = { max_size { MB | GB | TB } | UNLIMITED } ]
    [, MAX_ROLLOVER_FILES = integer]
    [, RESERVE_DISK_SPACE = { ON | OFF } ]
}

<audit_options>::=
{
    [ QUEUE_DELAY = integer ]
    [, ON_FAILURE = CONTINUE | SHUTDOWN ]
    [, AUDIT_GUID = uniqueidentifier ]
}
```

表18-21描述了这个命令的参数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表18-21 CREATE SERVER AUDIT的参数

参 数	描 述
audit_name	这个参数指定了服务器审核对象的用户定义名称
FILE (<file_options> [, ...n])] \|	这个参数指定服务器审核对象将事件写入文件中
APPLICATION_LOG	这个参数指定服务器审核对象将事件写入Windows应用程序事件日志中
SECURITY_LOG	这个参数指定服务器审核对象将事件写入Windows安全事件日志中
FILEPATH	如果选定的是FILE，这个参数指定审核日志操作系统文件路径
MAXSIZE	如果选定的是FILE，这个参数以MB、GB或TB定义最大文件大小。也可以指定为UNLIMITED
MAX_ROLLOVER_FILES	如果选定的是FILE，这个参数指定了文件在文件系统中保留的最大数量，当指定为0时，则没有限制
RESERVE_DISK_SPACE	这个参数使用ON或OFF值。当启用时，这个选项保留MAXSIZE指定的磁盘空间
QUEUE_DELAY	这个值指定在审核过程操作之前可以延迟的毫秒数，最小值和默认值为1000毫秒
ON_FAILURE	这个参数使用CONTINUE或SHUTDOWN值。如果指定SHUTDOWN，当不能写入目标时SQL实例将会关闭
AUDIT_GUID	这个选项使用服务器审核对象的唯一标识符。如果恢复包含数据库审核规范的数据库，则新的SQL实例的对象将会孤立，除非以匹配的GUID重建原始服务器审核对象

530

在这个技巧中，将创建新的配置为写入本地文件目录的服务器审核对象。指定每个日志最大为500兆字节，最多翻转10个文件。我不会保留磁盘空间，然后将队列延迟指定为1秒（1 000毫秒）。如果审核写入失败，不会关闭SQL Server实例：

```
USE master
GO

CREATE SERVER AUDIT Caesar_Augustus_Server_Audit
TO FILE
     ( FILEPATH = 'C:\Apress\',
       MAXSIZE = 500 MB,
       MAX_ROLLOVER_FILES = 10,
       RESERVE_DISK_SPACE = OFF)
WITH ( QUEUE_DELAY = 1000,
       ON_FAILURE = CONTINUE)
```

可以检查sys.server_audits目录视图来检验新服务器审核对象的配置：

```
SELECT audit_id,
       type_desc,
       on_failure_desc,
       queue_delay,
       is_state_enabled
FROM sys.server_audits
```

这个查询返回：

```
audit_id    type_desc   on_failure_desc  queue_delay    is_state_enabled
65536       FILE        CONTINUE         1000           0
```

正如在sys.server_audits目录视图的is_state_enabled列中看到的，服务器审核对象创建后，默认情况下是禁用状态。稍后，将在18.7.4节演示如何启用它，但与此同时，在我定义关联到服务器审

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

核对象的服务器和数据库审核规范之前，服务器审核对象都是禁用的。

为了查看刚创建的服务器审核对象的文件配置的详细信息，可以查询sys.server_file_audits目录视图：

531

```
SELECT name,
       log_file_path,
       log_file_name,
       max_rollover_files,
       max_file_size
FROM sys.server_file_audits
```

这个查询返回如下结果集（为了更好的显示，下面重排了格式）：

Column	Result
name	Caesar_Augustus_Server_Audit
log_file_path	C:\Apress\
log_file_name	Caesar_Augustus_Server_Audit_1F55EE1E-1BD3-4112-B108-F453330AF279.sqlaudit
max_rollover_files	10
max_file_size	500

解析

这个技巧演示了如何创建定义收集的审核事件目标去向的服务器审核对象。这是设置审核过程的第一步。下面将一点点地讲解代码，在第一行中指定了服务器审核对象的名称：

```
CREATE SERVER AUDIT Caesar_Augustus_Server_Audit
```

由于要将收集的审核事件目标保存到文件中，所以指定TO FILE：

```
TO FILE
```

下一步，指定了将要写入的审核文件的文件路径（由于是翻转文件，每个文件都是动态命名的——因此我只使用路径而不使用实际的文件名）：

```
( FILEPATH = 'C:\Apress\',
```

然后指定每个审核文件的最大字节和最多翻转文件的次数：

```
MAXSIZE = 500 MB,
MAX_ROLLOVER_FILES = 10,
```

还选择不保留空间（为了更好地实践，你应该将审核文件写入能确定有足够空间专用卷或的LUN中）：

```
RESERVE_DISK_SPACE = OFF)
```

最后，指定队列延迟为默认的1 000毫秒（1秒），如果写入目标文件失败，SQL Server实例将会继续运行（对于必须捕获的事件的关键任务审核，在事件写入目标文件发生问题时你可能会考虑关闭SQL实例）：

```
WITH ( QUEUE_DELAY = 1000,
       ON_FAILURE = CONTINUE)
```

在创建新的服务器审核对象之后，使用sys.server_audits目录视图来验证主服务器审核对象设置，使用sys.server_file_audits目录视图验证文件选项。

在下一个技巧中，将演示如何捕获在这个技巧中创建的服务器审核对象的SQL实例范围的事件。

532

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

18.7.2　捕获SQL实例范围的事件

使用服务器审核规范来定义哪些SQL实例范围事件将被捕获到服务器审核对象。可以使用CREATE SERVER AUDIT SPECIFICATION命令执行这个操作，语法如下：

```
CREATE SERVER AUDIT SPECIFICATION audit_specification_name
FOR SERVER AUDIT audit_name
{
    { ADD ( { audit_action_group_name } )
        } [, ...n]
     [ WITH ( STATE = { ON | OFF } ) ]
}
```

表18-22描述了这个命令的参数。

表18-22　CREATE SERVER AUDIT SPECIFICATION的参数

参　数	描　述
audit_specification_name	这个参数指定了服务器审核规范对象的用户定义名称
audit_name	这个参数定义了既有服务器审核对象（目标文件或事件日志）的名称
audit_action_group_name	这个参数表示SQL实例范围操作组的名称。你可以查询sys.dm_audit_actions目录视图来获取可审核的操作组列表
STATE	这个参数使用ON或OFF值。设置为ON时，开始收集记录

在这个技巧中，创建新的捕获3个不同审核操作组的服务器审核规范。要确定能够使用哪些审核组，可以查询sys.dm_audit_actions系统目录视图：

```
SELECT name
FROM sys.dm_audit_actions
WHERE class_desc = 'SERVER' AND
      configuration_level = 'Group'
ORDER BY name
```

这个查询返回如下部分结果：

```
name
APPLICATION_ROLE_CHANGE_PASSWORD_GROUP
AUDIT_CHANGE_GROUP
BACKUP_RESTORE_GROUP
BROKER_LOGIN_GROUP
DATABASE_CHANGE_GROUP
DATABASE_MIRRORING_LOGIN_GROUP
DATABASE_OBJECT_ACCESS_GROUP
...
DBCC_GROUP
FAILED_LOGIN_GROUP
LOGIN_CHANGE_PASSWORD_GROUP
LOGOUT_GROUP
...
SERVER_OBJECT_PERMISSION_CHANGE_GROUP
SERVER_OPERATION_GROUP
SERVER_PERMISSION_CHANGE_GROUP
SERVER_PRINCIPAL_CHANGE_GROUP
SERVER_PRINCIPAL_IMPERSONATION_GROUP
SERVER_ROLE_MEMBER_CHANGE_GROUP
SERVER_STATE_CHANGE_GROUP
SUCCESSFUL_LOGIN_GROUP
TRACE_CHANGE_GROUP
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在这个技巧中，希望跟踪所有DBCC的执行、BACKUP操作的执行或服务器角色成员关系的执行：

```
CREATE SERVER AUDIT SPECIFICATION Caesar_Augustus_Server_Audit_Spec
FOR SERVER AUDIT Caesar_Augustus_Server_Audit
   ADD (SERVER_ROLE_MEMBER_CHANGE_GROUP),
   ADD (DBCC_GROUP),
   ADD (BACKUP_RESTORE_GROUP)
WITH (STATE = ON)
```

创建服务器审核规范之后，就可以通过查询sys.server_audit_specifications目录视图来验证设置：

```
SELECT server_specification_id,
       name,
       is_state_enabled
FROM sys.server_audit_specifications
```

这个查询返回：

```
server_specification_id   name                                  is_state_enabled
65538                     Caesar_Augustus_Server_Audit_Spec     1
```

也可以通过查询sys.server_audit_specification_details目录视图来查询这个规范的细节（使用从前面查询返回的服务器规范ID来限定后面的结果集）：

```
SELECT server_specification_id,
       audit_action_name
FROM sys.server_audit_specification_details
WHERE server_specification_id = 65538
```

这个查询返回：

```
server_specification_id       audit_action_name
65538                         SERVER_ROLE_MEMBER_CHANGE_GROUP
65538                         BACKUP_RESTORE_GROUP
65538                         DBCC_GROUP
```

由于还没有启用服务器审核对象（Caesar_Augustus_Server_Audit），完整的审核机制尚未完成。在启用服务器审核对象之前，还会添加数据库审核规范对象，然后将演示实际的审核事件捕获以及如何查询审核日志。

解析

在这个技巧中，演示了如何创建定义了捕获哪些SQL实例范围的事件，并保存到指定服务器审核对象目标（在这个例子中，是在C:\Apress下的文件）的服务器审核规范。

在这个技巧中，首先通过查询sys.dm_audit_actions目录视图获取可以从SQL Server实例中选出审核的操作组。sys.dm_audit_actions目录视图实际包含所有审核操作的行——包括SQL实例和数据库范围。因此在查询的WHERE子句中，指定了审核操作的类应为SERVER，以及配置级别应为组（在下一技巧中演示非组操作级别的配置级别）：

```
...
WHERE class_desc = 'SERVER' AND
      configuration_level = 'Group'
...
```

接着，使用CREATE SERVER AUDIT SPECIFICATION命令来定义要跟踪哪个操作组。代码的第一行指

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

定了新服务器审核规范的名称：

```
CREATE SERVER AUDIT SPECIFICATION Caesar_Augustus_Server_Audit_Spec
```

下一行代码指定了事件收集的目标，即服务器审核对象的名称：

```
FOR SERVER AUDIT Caesar_Augustus_Server_Audit
```

在这之后，指定想要捕获的每一个操作组：

```
ADD (SERVER_ROLE_MEMBER_CHANGE_GROUP),
ADD (DBCC_GROUP),
ADD (BACKUP_RESTORE_GROUP)
```

最后，指定服务器审核规范的状态为在创建时启用：

```
WITH (STATE = ON)
```

在下一个技巧中，将演示如何创建捕获数据库范围事件的数据库审核规范。创建了所有规范之后，然后演示实际对操作的捕获，并展示如何读取服务器审核日志。

18.7.3　捕获数据库范围的事件

数据库审核规范用来定义哪些数据库范围的事件将被捕获到服务器审核对象。可以使用CREATE DATABASE AUDIT SPECIFICATION命令来执行这个操作，部分语法如下（这里没有展示操作规范的语法——但我会在技巧中演示）：

```
CREATE DATABASE AUDIT SPECIFICATION audit_specification_name
{
    [ FOR SERVER AUDIT audit_name ]
    [ { ADD (
            { <audit_action_specification> | audit_action_group_name }
                )
      } [, ...n] ]
    [ WITH ( STATE = { ON | OFF } ) ]
}
```

535

表18-23描述了这个命令的参数。

表18-23　CREATE DATABASE AUDIT SPECIFICATION的参数

参　数	描　述
audit_specification_name	这个参数指定了数据库审核规范对象的用户定义名称
audit_name	这个参数定义了既有的服务器审核对象（目标文件或事件日志）名称
audit_action_specification	这个参数表示可审核数据库范围操作的名称。你可以查询sys.dm_audit_actions目录视图列出可审核数据库范围操作
audit_action_group_name	这个参数定义了数据库范围操作组的名称。你可以查询sys.dm_audit_actions目录视图列出可审核操作组
STATE	这个参数可设为ON或OFF值。当指定为ON时，开始收集记录

在这个技巧中，将创建捕获审核操作组和审核事件的新数据库审核规范。审核操作组是数据库范围操作相关的组，审核事件是一个事件。例如，可以通过执行如下代码查询sys.dm_audit_actions系统目录视图，来查看对象的安全对象范围（例如，表、视图、存储过程和函数）的特定审核事件：

```
SELECT name
FROM sys.dm_audit_actions
WHERE configuration_level = 'Action' AND
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
      class_desc = 'OBJECT'
ORDER BY name
```

这个查询返回原子事件结果集，且这些对象的安全对象范围可被审核：

```
name
DELETE
EXECUTE
INSERT
RECEIVE
REFERENCES
SELECT
UPDATE
```

还可以查询sys.dm_audit_actions系统目录视图在数据库范围查看审核操作组：

```
SELECT name
FROM sys.dm_audit_actions
WHERE configuration_level = 'Group' AND
      class_desc = 'DATABASE'
ORDER BY name
```

这个查询返回如下结果（部分）：

```
name
APPLICATION_ROLE_CHANGE_PASSWORD_GROUP
AUDIT_CHANGE_GROUP
BACKUP_RESTORE_GROUP
CONNECT
DATABASE_CHANGE_GROUP
DATABASE_OBJECT_ACCESS_GROUP
...
DBCC_GROUP
SCHEMA_OBJECT_ACCESS_GROUP
SCHEMA_OBJECT_CHANGE_GROUP
SCHEMA_OBJECT_OWNERSHIP_CHANGE_GROUP
SCHEMA_OBJECT_PERMISSION_CHANGE_GROUP
```

在这个技巧中，想要跟踪所有数据库用户对HumanResources.Department表执行的所有INSERT、UPDATE或DELETE操作。还要跟踪AdventureWorks数据库中使用的所有模拟行为（例如，使用EXECUTE AS命令）：

```
USE AdventureWorks
GO

CREATE DATABASE AUDIT SPECIFICATION AdventureWorks_DB_Spec
FOR SERVER AUDIT Caesar_Augustus_Server_Audit
    ADD (DATABASE_PRINCIPAL_IMPERSONATION_GROUP),
    ADD (INSERT, UPDATE, DELETE
         ON HumanResources.Department
         BY public)
WITH (STATE = ON)
GO
```

可以通过查询sys.database_audit_specifications系统目录视图验证数据库审核规范的设置：

```
SELECT database_specification_id,name,is_state_enabled
FROM sys.database_audit_specifications
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回：

```
database_specification_id       name                        is_state_enabled
65538                           AdventureWorks_DB_Spec      1
```

可以查询sys.database_audit_specification_details系统目录视图来详细查看正在审核的新数据库审核规范（我会在“解析”部分一点点地讲解逻辑）：

```
SELECT audit_action_name,
       class_desc,
       is_group,
       CASE
           WHEN major_id > 0 THEN OBJECT_NAME(major_id, DB_ID())
           ELSE 'N/A'
       END ObjectNM
FROM sys.database_audit_specification_details
WHERE database_specification_id = 65538
```

537

这个查询返回：

```
audit_action_name                          class_desc          is_group   ObjectNM
DATABASE_PRINCIPAL_IMPERSONATION_GROUP     DATABASE            1          N/A
DELETE                                     OBJECT_OR_COLUMN    0          Department
INSERT                                     OBJECT_OR_COLUMN    0          Department
UPDATE                                     OBJECT_OR_COLUMN    0          Department
```

尽管启用了数据库审核规范，仍然没有启用整个服务器审核对象。在下一技巧中会演示，你也会学习到如何从二进制文件中查询已捕获的审核数据。

解析

在这个技巧中，演示了如何创建指定哪些数据库范围事件将被捕获到服务器审核对象的数据库审核规范。使用CREATE DATABASE AUDIT SPECIFICATION命令来执行这个操作。首先，更改要审核数据库的上下文（因为这是数据库范围对象）：

```
USE AdventureWorks
GO
```

CREATE DATABASE AUDIT SPECIFICATION命令的第一行指定了用户定义的名称，随后是要保存数据范围事件的对服务器审核对象的引用：

```
CREATE DATABASE AUDIT SPECIFICATION AdventureWorks_DB_Spec
FOR SERVER AUDIT Caesar_Augustus_Server_Audit
```

在这之后，使用ADD关键字，随后是右括号，定义要监控的审核操作组，然后是左括号和一个逗号（因为我计划定义监控多个操作）：

```
ADD (DATABASE_PRINCIPAL_IMPERSONATION_GROUP),
```

下一步，再次使用ADD关键字，随后是希望监控HumanResources.Department表的3个操作：

```
ADD (INSERT, UPDATE, DELETE
    ON HumanResources.Department
```

对象范围的操作需要一个到希望审核操作的数据库主体的引用。在这个示例中，希望通过所有的数据库主体来查看操作。由于所有数据库主体默认为public的成员，这里指定：

```
BY public)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在这之后，使用WITH关键字，随后是STATE参数，并将它设置为启用状态：

```
WITH (STATE = ON)
GO
```

然后使用sys.database_audit_specifications查看新数据库审核规范的基本信息。查询sys.database_audit_specification_details系统目录视图列出数据库审核规范捕获的事件，在前三行代码中，看到审核操作的名称、类描述和is_group段，它指定审核操作是否是审核操作组还是单独的事件：

538


```
SELECT audit_action_name,
       class_desc,
       is_group,
```

使用CASE语句来计算major_id列。如果major_id是非零值，它表示审核操作行是数据库对象，所以我使用OBJECT_NAME函数来提供对象的名称：

```
CASE
     WHEN major_id > 0 THEN OBJECT_NAME(major_id, DB_ID())
     ELSE 'N/A'
END ObjectNM
```

在SELECT的最后两行，指定了目录视图的名称，并指定了数据库规范ID（如果你为数据库定义了多个数据库审核规范这是重要的——也是允许的）：

```
FROM sys.database_audit_specification_details
WHERE database_specification_id = 65538
```

现在已经定义完服务器审核对象、服务器审核规范和数据库审核规范，在下一技巧中，将演示启用服务器审核对象并创建一些可审核的活动，然后展示如何查询捕获到的审核数据。

18.7.4 查询捕获的审核数据

前一技巧进行了一个实际的关于SQL Server 2008审核能力的演示。在这个技巧中，首先将启用前面创建的服务器审核对象。回忆一下我曾定义这个服务器审核对象在C:\Apress文件夹下写入二进制文件。

使用ALTER SERVER AUDIT命令并配置STATE选项来启用审核：

```
USE master
GO

ALTER SERVER AUDIT [Caesar_Augustus_Server_Audit] WITH (STATE = ON)
```

为了演示审核收集的过程，现在我将在SQL Server范围和AdventureWorks数据库中执行几个操作。我在每个语句组之前添加了注释，这样你就可以理解要演示的操作了：

```
USE master
GO

-- 创建新登录名（不会审核它，但在技巧中会用到）
CREATE LOGIN TestAudit WITH PASSWORD = 'C83D7F50-9B9E'

-- 增加服务器角色 bulkadmin
EXEC sp_addsrvrolemember 'TestAudit', 'bulkadmin'
GO

-- 备份 AdventureWorks 数据库
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
BACKUP DATABASE AdventureWorks
TO DISK = 'C:\Apress\Example_AW.BAK'
GO

-- 在AdventureWorks数据库中执行DBCC操作
DBCC CHECKDB('AdventureWorks')
GO

-- 执行一些AdventureWorks 操作
USE AdventureWorks
GO

-- 创建新用户然后在这个用户的上下文中执行
CREATE USER TestAudit FROM LOGIN TestAudit

EXECUTE AS USER = 'TestAudit'

-- 返回到本人(在这个例子中是sysadmin角色的登录名)
REVERT

-- 对HumanResources.Department表执行INSERT、UPDATE和DELETE操作

INSERT HumanResources.Department
(Name, GroupName)
VALUES('Traffic', 'Advertising')

UPDATE HumanResources.Department
SET Name = 'Media Planning'
WHERE Name = 'Traffic'

DELETE HumanResources.Department
WHERE Name = 'Media Planning'
```

539

现在已经执行了几个被前面创建的服务器审核规范和数据库审核规范覆盖的事件，可以使用fn_get_audit_file表值函数来查看服务器审核二进制文件的内容。这个函数的语法如下：

```
fn_get_audit_file
( file_pattern,
 {default | initial_file_name | NULL },
 {default | audit_file_offset | NULL } )
```

表18-24描述了这个命令的参数。

表18-24 fn_get_audit_file的参数

参 数	描 述
file_pattern	指定要读取的审核文件的位置。路径你可以使用盘符或网络共享，也可以使用一个星号（*）通配符来指定多个文件
{default \| initial_file_name \| NULL }	指定你要开始读取的特定文件的名称或路径。default和NULL意义相似，表示没有选取初始文件名
{default \| audit_file_offset \| NULL }	指定初始文件的缓冲区偏移量（当选择初始文件时）。defaul和NULL意义相似，表示没有选取审核文件的偏移量

540

第一次调用fn_get_audit_file函数，查找所有发生在服务器角色成员上的变更。注意，使用sys.dm_audit_actions目录视图将活动ID翻译为实际活动的事件名称（可以使用这个视图来查找要筛选哪些事件名称）：

```
SELECT af.event_time,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
       af.succeeded,
       af.target_server_principal_name,
       object_name
FROM fn_get_audit_file
      ('C:\Apress\Caesar_Augustus_Server_Audit_*',
       default, default) af
INNER JOIN sys.dm_audit_actions aa ON
       af.action_id = aa.action_id
WHERE aa.name = 'ADD MEMBER' AND
      aa.class_desc = 'SERVER ROLE'
```

这个查询返回事件时间、成功标记、服务器主体名称和服务器角色名称：

```
event_time              succeeded  target_server_principal_name  object_name
2008-09-02 15:06:54.702 1          TestAudit                    bulkadmin
```

在下一个示例中，将讲解HumanResources.Department表的删除事件：

```
SELECT af.event_time,
       af.database_principal_name
FROM fn_get_audit_file
      ('C:\Apress\Caesar_Augustus_Server_Audit_*',
       default, default) af
INNER JOIN sys.dm_audit_actions aa ON
       af.action_id = aa.action_id
WHERE aa.name = 'DELETE' AND
      aa.class_desc = 'OBJECT' AND
      af.schema_name = 'HumanResources' AND
      af.object_name = 'Department
```

这个查询返回：

```
event_time              database_principal_name
2008-09-02 15:13:24.542 dbo
```

fn_get_audit_file函数还会暴露出恰当的出现在实例化事件中的SQL语句。随后的查询演示了捕获实际审核事件使用的BACKUP DATABASE文本：

```
SELECT event_time, statement
FROM fn_get_audit_file
      ('C:\Apress\Caesar_Augustus_Server_Audit_*',
       default, default) af
INNER JOIN sys.dm_audit_actions aa ON
       af.action_id = aa.action_id
WHERE aa.name = 'BACKUP' AND
      aa.class_desc = 'DATABASE'
```

这个查询返回事件时间和相应的BACKUP语句文本：

```
event_time              statement
2008-02-02 15:07:29.482 BACKUP DATABASE AdventureWorks TO DISK =
                        'C:\Apress\Example_AW.BAK'
```

本技巧的最后一个查询演示了查询所有不同的事件和相关联的执行它的数据库主体，以及目标服务器主体名称（适用时）或目标对象名称：

```
SELECT DISTINCT
       aa.name,
       database_principal_name,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
       target_server_principal_name,
       object_name
FROM fn_get_audit_file
      ('C:\Apress\Caesar_Augustus_Server_Audit_*',
        default, default) af
INNER JOIN sys.dm_audit_actions aa ON
       af.action_id = aa.action_id
```

这个查询返回之前执行的并定义在服务器和数据库审核规范中的各种事件。它还包括默认的审核事件——例如，AUDIT SESSION CHANGED：

name	database_principal_name	target_server_principal_name	object_name
ADD MEMBER	dbo	TestAudit	bulkadmin
AUDIT SESSION CHANGED			
BACKUP	dbo		AdventureWorks
DBCC	dbo		
DELETE	dbo		EmployeePayHistory
IMPERSONATE	dbo		TestAudit
INSERT	dbo		EmployeePayHistory
UPDATE	dbo		EmployeePayHistory

解析

在这个技巧中，首先使用ALTER SERVER AUDIT命令启用整个服务器审核对象。在这之后，执行了一些SQL实例范围和数据库范围的活动——关注已经定义的绑定到Caesar_Augustus_Server_Audit审核的服务器和数据库审核规范中捕获的事件。在这之后，演示了如何使用fn_get_audit_file函数从在C:\Apress目录下的二进制文件中获取数据。

注意 也可以定义服务器审核对象将事件写入Windows应用程序事件日志或Windows安全事件日志中，在这个情况下我不会使用fn_get_audit_file来返回数据，因为这个函数只能应用于二进制文件格式。

每个fn_get_audit_file的查询都联结到sys.dm_audit_actions对象以指定审核操作名称，还有类描述（依赖于操作）。例如：

542

```
...
FROM fn_get_audit_file
      ('C:\Apress\Caesar_Augustus_Server_Audit_*',
        default, default) af
INNER JOIN sys.dm_audit_actions aa ON
       af.action_id = aa.action_id
WHERE aa.name = 'ADD MEMBER' AND
      aa.class_desc = 'SERVER ROLE'
...
```

在下一个也是本章最后一个技巧中，将演示如何管理、修改和移除审核对象。

18.7.5 管理、修改和移除审核对象

这个技巧将演示如何添加和从既有的服务器和数据库审核规范中移除操作、禁用服务器和数据库审核规范、修改服务器审核对象，以及从SQL实例和关联的数据库中移除审核对象。

使用ALTER SERVER AUDIT SPECIFICATION命令修改既有的服务器审核规范。在第一个查询演示中，从前面创建的服务器审核规范中移除一个审核操作类型，再增加一个新的审核操作。

在修改服务器审核规范之前，必须先禁用它：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
USE master
GO

ALTER SERVER AUDIT SPECIFICATION [Caesar_Augustus_Server_Audit_Spec]
WITH (STATE = OFF)
```

下一步，将删除一个审核操作：

```
ALTER SERVER AUDIT SPECIFICATION [Caesar_Augustus_Server_Audit_Spec]
DROP (BACKUP_RESTORE_GROUP)
```

现在，将演示增加一个新的审核操作组到既有的服务器审核规范中：

```
ALTER SERVER AUDIT SPECIFICATION [Caesar_Augustus_Server_Audit_Spec]
ADD (LOGIN_CHANGE_PASSWORD_GROUP)
```

必须重新启用服务器审核规范，才可以使这些改变生效并继续审核：

```
ALTER SERVER AUDIT SPECIFICATION [Caesar_Augustus_Server_Audit_Spec]
WITH (STATE = ON)
```

必须使用ALTER DATABASE AUDIT SPECIFICATION命令来修改数据库审核规范中的审核操作。与服务器审核规范相似，数据库审核规范在修改之前必须处于禁用状态：

```
USE AdventureWorks
GO

ALTER DATABASE AUDIT SPECIFICATION [AdventureWorks_DB_Spec]
WITH (STATE = OFF)
```

下一个查询演示从前面创建的数据库审核规范中移除既有审核事件：

```
ALTER DATABASE AUDIT SPECIFICATION [AdventureWorks_DB_Spec]
DROP (INSERT ON [HumanResources].[Department] BY public)
```

下一步，演示了如何增加新的审核事件到既有的数据库审核规范中：

```
ALTER DATABASE AUDIT SPECIFICATION [AdventureWorks_DB_Spec]
ADD (DATABASE_ROLE_MEMBER_CHANGE_GROUP)
```

为了使修改生效，需要重新启用数据库审核规范：

```
ALTER DATABASE AUDIT SPECIFICATION [AdventureWorks_DB_Spec]
WITH (STATE = ON)
```

使用ALTER SERVER AUDIT命令来修改服务器审核对象本身。与服务器和数据库审核规范对象相似，在修改生效之前需要禁用服务器审核对象。在下一个示例中，演示禁用服务器审核，修改日志写入的目标到Windows应用程序事件日志中，然后重新启用它：

```
USE master
GO

ALTER SERVER AUDIT [Caesar_Augustus_Server_Audit]
WITH (STATE = OFF)

ALTER SERVER AUDIT [Caesar_Augustus_Server_Audit]
TO APPLICATION_LOG

ALTER SERVER AUDIT [Caesar_Augustus_Server_Audit]
WITH (STATE = ON)
```

修改了目标之后，现在审核事件就会保存到Windows应用程序事件日志中。例如，如果再次使用DBCC CHECKDB命令，就会在Windows应用程序事件日志中以事件ID 33205反映出这个操作。下面是

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Windows应用程序事件日志项的示例：

```
Audit event: event_time:2008-09-02 18:17:49.4704464
sequence_number:1
action_id:DBCC
succeeded:true
permission_bitmask:0
is_column_permission:false
session_id:57
server_principal_id:263
database_principal_id:1
target_server_principal_id:0
target_database_principal_id:0
object_id:0
class_type:DB
session_server_principal_name:CAESAR\Administrator
server_principal_name:CAESAR\Administrator
server_principal_sid:0105000000000005150000006bb13b36a981eb9a2b3859a8f4010000
database_principal_name:dbo
target_server_principal_name:
target_server_principal_sid:
target_database_principal_name:
server_instance_name:CAESAR\AUGUSTUS
database_name:AdventureWorks
schema_name:
object_name:
statement:DBCC CHECKDB('AdventureWorks')
additional_information:
```

544

首先需要禁用数据库审核规范，然后使用DROP DATABASE AUDIT SPECIFICATION命令移除数据库审核规范，如下所示：

```
USE AdventureWorks
GO

ALTER DATABASE AUDIT SPECIFICATION [AdventureWorks_DB_Spec]
WITH (STATE = OFF)

DROP DATABASE AUDIT SPECIFICATION [AdventureWorks_DB_Spec]
```

首先需要禁用服务器审核规范，然后使用DROP SERVER AUDIT SPECIFICATION命令移除服务器审核规范：

```
USE master
GO

ALTER SERVER AUDIT SPECIFICATION [Caesar_Augustus_Server_Audit_Spec]
WITH (STATE = OFF)

DROP SERVER AUDIT SPECIFICATION [Caesar_Augustus_Server_Audit_Spec]
```

最后，需要先禁用服务器审核对象，然后使用DROP SERVER AUDIT命令删除服务器审核对象，如下所示：

```
ALTER SERVER AUDIT [Caesar_Augustus_Server_Audit] WITH (STATE = OFF)

DROP SERVER AUDIT [Caesar_Augustus_Server_Audit]
```

在移除服务器审核对象之后，所有为审核创建的二进制日志文件仍将保留。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

这个技巧演示了几个用于管理审核对象的命令。对于所有既有的审核对象，需要在使修改生效前处于禁用状态。使用ALTER SERVER AUDIT SPECIFICATION命令增加审核事件和从服务器审核规范中移除审核事件，使用DROP SERVER AUDIT SPECIFICATION命令从SQL Server实例中移除规范的定义。

使用ALTER DATABASE AUDIT SPECIFICATION命令增加审核事件和从数据库审核规范移除审核事件，使用DROP DATABASE AUDIT SPECIFICATION命令从用户数据库中移除规范的定义。使用ALTER SERVER AUDIT命令修改既有服务器审核对象——将记录日志的目标方法从二进制文件更改为Windows应用程序事件日志。最后，使用DROP SERVER AUDIT命令从SQL Server实例中移除服务器审核对象。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 19 章 加　密

在SQL Server 2005和SQL Server 2008之前，如果希望加密敏感数据，比如财务信息、工资或个人识别号码，必须依靠外部应用程序或算法。SQL Server 2005引入内建数据加密的能力，使用证书、密钥和系统函数的组合来完成。

与通过证书授权实现的数字证书相似，SQL Server证书包括公钥和私钥这一对密钥，它们用来加密和解密数据。SQL Server也拥有创建非对称密钥和对称密钥对象的能力。非对称密钥（asymmetric key）与证书相似，公钥用来加密数据库，私钥用来解密数据。非对称密钥和证书都提供了强大的加密强度，但在完成复杂的加密／解密过程中具有更多的性能开销。更适合于对大量数据进行加密，且具有较低性能开销的解决方案是对称密钥（symmetric key），它是对相同数据进行加密和解密的一个密钥。

SQL Server允许将这些加密能力放到加密层次结构中。当安装了SQL Server后，在数据库master中创建名为服务主密钥（Service Master Key）的服务器级别证书，并将其默认绑定到SQL Server服务账户登录名。服务主密钥用来加密所有其他数据库证书和创建在SQL Server实例中的密钥。另外，你也可以在用户数据库中创建数据库主密钥（DataBase Master Key），它可以用来加密数据库证书和密钥。

在SQL Server 2008中，微软引入了透明数据加密（TDE），它对整个数据库进行加密，而不需要修改任何访问它的应用程序。数据、日志文件和相关的数据库备份都是加密的。假如数据库被偷，如果没有数据库加密密钥（DEK）是不能访问数据的。在本章将演示如何启用这个新功能。

提示　SQL Server 2008还引入了对可扩展密钥管理（EKM）的支持，意思是SQL Server可以使用硬件安全模块（HSM）来存储和管理加密密钥。HSM可以减少数据和实际的加密密钥耦合。

我们一开始先讨论，然后演示如何不使用证书和密钥对数据进行加密。

19.1　通过通行短语加密

对于一个不涉及证书或密钥的应急的数据加密，可以直接基于用户提供的密码加密／解密数据。通行短语是允许存在空格的密码。这个通行短语不会存储在数据库中。这是有利的，因为这意味着内部密码不会被使用存储的系统数据“破解”。因为密码可以包含空格，所以可以创建一个长的、易于记忆的句子来用作加密和解密敏感数据。

在下一个技巧中我将演示如何使用通行短语函数加密和解密数据。

使用函数通过通行短语加密

可以调用函数EncryptByPassPhrase，使用用户提供的通行短语来加密数据。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

语法如下：

```
EncryptByPassPhrase(
 { ' passphrase ' | @passphrase }
, { ' cleartext ' | @cleartext }
[ , { add_authenticator | @add_authenticator }
, { authenticator | @authenticator } ]     )
```

表19-1描述了这个命令的参数。

表19-1 EncryptByPassPhrase的参数

参 数	描 述
' passphrase ' \| @passphrase	用来加密数据的通行短语
'cleartext' \| @cleartext	要加密的文本
add_authenticator \| @add_authenticator	决定指示是否将验证器与明文一起加密的布尔值（1或0）
authenticator \| @authenticator	用于派生验证器的数据

使用DecryptByPassPhrase函数来解密被加密的值，它使用与EncryptByPassPhrase相同的参数，只是用密文取代了明文：

```
DecryptByPassPhrase(
{ ' passphrase ' | @passphrase }
, { ' ciphertext ' | @ciphertext }
[ , { add_authenticator | @add_authenticator }
, { authenticator | @authenticator } ] )
```

在这个技巧中，使用通行短语加密字符串my secure secret text：

```
--这个技巧中使用了用来存储加密数据的表
CREATE TABLE #SecretInfo
(Secret varbinary(8000) NOT NULL)
GO

INSERT #SecretInfo
(Secret)
SELECT   EncryptByPassPhrase(
         'My Password Used To Encrypt This String in 2008.',
         'This is the text I need to secure.')

SELECT Secret
FROM #SecretInfo
```

这个查询返回如下信息（你的结果可能会不同）：

548

```
0x0100000031AF7E0656FB1C3253AE708B4DB5F3F1EDEA48C832E5BE493E01655D8E7783D6C21E
2B94817636EAD39328D940B8BD4F9718081E6EB837BE
```

从表#SecretInfo中获取返回的varbinary值，可以使用相同的通行短语对文本进行解密（使用错误的通行短语会返回NULL值）：

```
SELECT CAST(DecryptByPassPhrase(
'My Password Used To Encrypt This String in 2008.',
Secret) as varchar(50))
FROM #SecretInfo
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
This is the text I need to secure.
```

解析

在这个技巧中，使用了临时表来保存函数EncryptByPassPhrase加密的输出。定义的列数据类型为varbinary（8000）（8 000是这个函数可以加密的最大长度）：

```
CREATE TABLE #SecretInfo
(Secret varbinary(8000) NOT NULL)
GO
```

下一步，使用INSERT...SELECT将新行插入到临时表中：

```
INSERT #SecretInfo
(Secret)
```

SELECT引用了函数EncryptByPassPhrase。第一个参数是用来加密字符串的实际密码（在这个例子中是整个句子）。第二个参数是要加密的字符串：

```
SELECT   EncryptByPassPhrase('My Password Used To Encrypt This String in 2008.',
         'This is the text I need to secure.')
```

下一步查询了插入的varbinary(8000)值，返回一个无意义的值：

```
SELECT Secret
FROM #SecretInfo
```

然后使用函数DecryptByPassPhrase来解密数据，将密码作为第一个参数（用来在第一个位置加密数据的原始的一个），并且引用临时表#SecretInfo的Secret列中的加密数据：

```
SELECT CAST(DecryptByPassPhrase(
'My Password Used To Encrypt This String in 2008.',
Secret) as varchar(50))
FROM #SecretInfo
```

使用通行短语加密函数加密数据，就不用担心sysadmin服务器角色成员读取数据（在本章中会看
549 到，服务器角色成员sysadmin拥有读取其他形式的加密数据的内在权限）。

假定没有将密码存储在表中或在任何模块（存储过程、触发器等）中使用密码，加密的数据将防止从数据库备份被偷窃或在SQL Server实例中的数据库渗透。如果通行短语没有适当地共享，数据就可以被解密。

19.2　主密钥

在SQL Server中的加密由层次结构形式进行处理以提供多级别的安全。SQL Server包含两个用于加密数据的密钥类型。*服务主密钥*（Service Master Key）位于层次结构的最顶端，并且在安装SQL Server时自动创建。服务主密钥也用于加密其下的数据库主密钥。*数据库主密钥*（Database Master Key）用来加密证书，以及非对称密钥和对称密钥。在这个部分，将讨论这两种不同类型的密钥：服务主密钥和数据库主密钥。

如前所述，服务主密钥位于SQL Server中加密层次结构的顶端，并且它用于加密系统数据、链接的服务器登录名以及数据库主密钥。在第一次通过SQL Server使用服务主密钥来加密证书、数据库主密钥或链接的服务器密码时，服务主密钥会自动生成，并且是使用SQL Server服务账户的Windows证书来生成它。如果必须改变SQL Server服务账户，微软建议使用SQL Server配置管理器，因为这个工具将执行生成新服务主密钥需要的合适的解密和加密方法，而且可以使加密层次结构保持完整。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

数据库主密钥处于加密层次结构中SQL Server安全性的其他层，它可以用来加密数据库证书和非对称密钥。所有数据库都可以只包含一个数据库主密钥，创建它时，通过服务主密钥对其加密。

创建非对称密钥（本章19.3节会介绍）时，可以决定在加密非对称密钥对的私钥时是否包含密码。如果不包含密码，将使用数据库主密钥来加密私钥。使用数据库主密钥来加密其他对象是一个很好的例子。

在下面一组技巧中，将演示如何管理这两种不同的密钥类型。

19.2.1 备份及还原服务主密钥

因为服务主密钥是SQL Server中的关键角色，所以在它损坏或被修改的情况下，将这个密钥备份到安全的地方是非常重要的。通过使用BACKUP SERVICE MASTER KEY命令来执行备份操作。

语法如下：

```
BACKUP SERVICE MASTER KEY TO FILE = 'path_to_file'
    ENCRYPTION BY PASSWORD = 'Password'
```

这个命令有两个参数：第一个参数是密钥备份将导出到的路径和文件名。第二个参数是用来加密包含密钥备份的文件的密码。在备份服务主密钥之后，应该将备份文件备份到磁带或从服务器中复制到安全的地方。

如果需要从SQL Server实例中的备份恢复服务主密钥，可以使用RESTORE SERVICE MASTER KEY命令。

语法如下：

```
RESTORE SERVICE MASTER KEY FROM FILE = 'path_to_file'
    DECRYPTION BY PASSWORD = 'password' [FORCE]
```

550

这个命令使用了备份文件的名称和加密密码。FORCE参数用来强制替换既有的服务主密钥，即便是在数据丢失的情况下（因此应该只在紧急的情况下，并且你能够承受丢失加密的数据不能被解密的情况下使用它）。

这个技巧演示了备份并且还原服务主密钥。

在第一个示例中，使用BACKUP SERVICE MASTER KEY命令将服务主密钥备份到C:\Apress\Recipes目录下的文件上：

```
BACKUP SERVICE MASTER KEY
TO FILE = 'C:\Apress\Recipes\SMK.bak'
ENCRYPTION BY PASSWORD = 'MakeItAGoodOne!1AB'
```

如下代码演示了从备份文件恢复服务主密钥：

```
RESTORE SERVICE MASTER KEY
FROM FILE = 'C:\Apress\Recipes\SMK.bak'
DECRYPTION BY PASSWORD = 'MakeItAGoodOne!1AB'
```

解析

在第一个示例中，将服务主密钥备份到文件。代码第二行指定了要将文件备份到的文件名：

```
BACKUP SERVICE MASTER KEY
TO FILE = 'C:\Apress\Recipes\SMK.bak'
```

代码的第三行指定了用来保护文件的密码（而且是执行还原所必需的）：

```
ENCRYPTION BY PASSWORD = 'MakeItAGoodOne!1AB'
```

在第二个示例中，执行了服务主密钥还原。代码的第二行指定了服务主密钥从哪个文件名进行还原：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
RESTORE SERVICE MASTER KEY
FROM FILE = 'C:\Apress\Recipes\SMK.bak'
```

代码第三行指定了用来保护和生成服务主密钥备份的密码：

```
DECRYPTION BY PASSWORD = 'MakeItAGoodOne!1AB'
```

如果你自己测试这个示例，且在服务主密钥不做任何实际的改变的情况下执行备份和还原操作，那么在执行RESTORE操作的过程中你将看到如下的消息：

```
新旧主密钥完全相同。不需要重新加密数据。
```

19.2.2 创建、重新生成以及删除数据库主密钥

当显式创建数据库主密钥时，它会通过自动地在数据库中加密新证书或非对称密钥增加一个额外的安全层，用于进一步保护加密的数据。

使用CREATE MASTER KEY命令创建数据库主密钥。语法如下：

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'password'
```

就像服务主密钥，数据库主密钥不需要显式的名称，并且只使用一个参数：数据库主密钥的密码。可以通过使用ALTER MASTER KEY命令重新生成数据库主密钥。重新生成数据库主密钥的语法如下：

551

```
ALTER MASTER KEY
[FORCE] REGENERATE WITH ENCRYPTION BY PASSWORD = 'password'
```

这个命令只有一个参数，重新生成密钥的密码。重新生成密钥解密通过密钥加密的所有对象，并且使用新的重新生成的密钥重新加密它们。如果在解密过程中出现错误（由于各种各样的原因不能解密数据），FORCE选项强制进行重新生成过程，但是*也带着使加密的数据变得不可用的危险*，记住这是很重要的。

使用DROP MASTER KEY命令彻底地删除数据库主密钥（不需要其他参数）。

例如：

```
USE BookStore
GO
DROP MASTER KEY
```

但是，如果它仍然用来加密其他数据库对象，就不可以删除数据库主密钥。

在第一个示例中，为数据库BookStore创建数据库主密钥：

```
IF NOT EXISTS (SELECT name
          FROM sys.databases
          WHERE name = 'BookStore')
BEGIN
     CREATE DATABASE BookStore
END
GO

USE BookStore
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '99a555ac-cf60-472b-9c1e-ed735ffbb089'
```

下一步，我将演示使用新密码重新生成数据库主密钥：

```
Use BookStore
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ALTER MASTER KEY
REGENERATE WITH ENCRYPTION BY PASSWORD = 'uglypassword1C3ED8CF'
```

最后，我将删除数据库主密钥（没有用它加密其他密钥时，我才可以完成删除操作）：

```
DROP MASTER KEY
```

解析

这个示例演示了在数据库BookStore中创建数据库主密钥。用户只需提供用来加密数据库主密钥加密的密码：

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '99a555ac-cf60-472b-9c1e-ed735ffbb089'
```

552

第二个示例也只需要一个用户提供的参数，即用来重新生成新数据库主密钥的密码：

```
ALTER MASTER KEY
REGENERATE WITH ENCRYPTION BY PASSWORD = 'uglypassword1C3ED8CF'
```

然后，使用如下命令删除数据库主密钥：

```
DROP MASTER KEY
```

19.2.3 备份及还原数据库主密钥

像服务主密钥一样，也可以使用BACKUP MASTER KEY命令将数据库主密钥备份到磁盘。语法如下：

```
BACKUP MASTER KEY TO FILE = 'path_to_file'
    ENCRYPTION BY PASSWORD = 'Password'
```

该命令用了两个参数，第一个是路径和文件名（数据库主密钥要备份到的地方），第二个是用来保护备份文件的密码。

要从文件备份还原数据库主密钥，可使用RESTORE MASTER KEY命令。语法如下：

```
RESTORE MASTER KEY FROM FILE = 'path_to_file'
    DECRYPTION BY PASSWORD = 'password'
    ENCRYPTION BY PASSWORD = 'password'
    [FORCE]
```

这个命令使用了文件名和路径、用来解密备份文件的密码以及加密新数据库主密钥的新密码。FORCE选项强制数据库主密钥还原，即便数据库中所有依赖的加密数据不可以使用新密钥进行重加密。因为依赖的加密的数据不可以被解密，这意味着它将变得不可用——因此使用该选项时要小心，并且作为最后的手段使用它！

在第一个示例中，我在BookStore数据库中创建了主密钥，然后将数据库主密钥备份到文件：

```
USE BookStore
GO

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'MagneticFields!'
GO

BACKUP MASTER KEY TO FILE = 'C:\Apress\Recipes\BookStore_Master_Key.BAK'
    ENCRYPTION BY PASSWORD = '4D280837!!!'
```

接着，我演示了从文件还原数据库主密钥：

```
RESTORE MASTER KEY FROM FILE = 'C:\Apress\Recipes\BookStore_Master_Key.BAK'
DECRYPTION BY PASSWORD = '4D280837!!!'
ENCRYPTION BY PASSWORD = 'MagneticFields!'
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

正如你在随后的技巧中看到的，数据库主密钥用来加密其他从属的加密对象。所以，在刚创建数据库主密钥后就立即备份是一个好主意。

在这个技巧中，我为用户数据库创建了数据库主密钥，然后将数据库主密钥备份到文件，在BACKUP
553 MASTER KEY命令的第一个参数中指定这个文件：

```
BACKUP MASTER KEY TO FILE = 'C:\Apress\Recipes\BookStore_Master_Key.BAK'
```

代码的第二行指定了用于加密备份文件的密码：

```
ENCRYPTION BY PASSWORD = '4D280837!!!'
```

第二个示例演示了从文件还原数据库主密钥。代码的第一行指定了备份文件的名称：

```
RESTORE MASTER KEY FROM FILE = 'C:\Apress\Recipes\BookStore_Master_Key.BAK'
```

第二行指定了用于原始加密备份文件的密码：

```
DECRYPTION BY PASSWORD = '4D280837!!!'
```

代码的第三行指定了用于加密还原后的数据库主密钥的密码：

```
ENCRYPTION BY PASSWORD = 'MagneticFields!'
```

如果在SQL Server实例之外测试这个示例，并且数据库主密钥在备份和还原之间没有改变，你将看到如下消息：

```
新旧主密钥完全相同。不需要重新加密数据。
```

19.2.4 从数据库主密钥删除服务主密钥加密

创建数据库主密钥时，默认使用两种方法对它加密：服务主密钥和在CREATE MASTER KEY命令中使用的密码。如果你不希望通过服务主密钥对数据库主密钥进行加密（这样如果拥有sysadmin权限的SQL Server登录名不知道数据库主密钥密码，它就不能访问加密的数据），你可以使用ALTER MASTER KEY的一个变种形式删除它。

部分语法如下：

```
ALTER MASTER KEY
ADD ENCRYPTION BY SERVICE MASTER KEY |
DROP ENCRYPTION BY SERVICE MASTER KEY
```

因为服务主密钥允许通过拥有合适权限（例如，sysadmin）的用户自动对数据库主密钥解密，所以如果你希望去修改它，一旦你通过服务主密钥删除加密，则必须使用新命令来访问数据库主密钥。这个命令是OPEN MASTER KEY，它的语法如下：

```
OPEN MASTER KEY DECRYPTION BY PASSWORD = 'password'
```

在CLOSE MASTER KEY命令中使用数据库主密钥（没有其他参数）来完成这个示例。

在这个示例中，从数据库BookStore中删除使用服务主密钥的加密：

```
USE BookStore
GO
```

554

```
ALTER MASTER KEY DROP ENCRYPTION BY SERVICE MASTER KEY
```

为了重新启用使用服务主密钥的加密，我们必须首先打开到数据库主密钥的访问，服务主密钥加

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

密重新被添加到数据库主密钥中，然后再一次关闭数据库主密钥：

```
OPEN MASTER KEY DECRYPTION BY PASSWORD = 'MagneticFields!'

ALTER MASTER KEY ADD ENCRYPTION BY SERVICE MASTER KEY

CLOSE MASTER KEY
```

一旦使用服务主密钥来加密数据库主密钥，数据库主密钥就不再需要显式地打开或关闭。

解析

这个技巧演示了使用ALTER MASTER KEY命令删除数据库主密钥上使用服务主密钥的加密：

```
ALTER MASTER KEY DROP ENCRYPTION BY SERVICE MASTER KEY
```

完成这个之后，数据库主密钥的所有修改都需要使用OPEN MASTER KEY命令进行密码访问。使用它是为了重新应用使用服务主密钥的加密：

```
OPEN MASTER KEY DECRYPTION BY PASSWORD = 'MagneticFieldS!'
```

然后使用ALTER MASTER KEY命令再一次将服务主密钥增加回来：

```
ALTER MASTER KEY ADD ENCRYPTION BY SERVICE MASTER KEY
```

完成ALTER MASTER KEY操作之后，关闭数据库主密钥：

```
CLOSE MASTER KEY
```

19.3 非对称密钥加密

非对称密钥包含数据库级的内部公钥和私钥，它可以用来加密和解密SQL Server数据库中的数据。非对称密钥可以从外部文件或程序集中导入，并且也可以在SQL Server数据库中生成。

不像证书（将在本章后面进行讨论），非对称密钥不可以备份到文件。这意味着如果在SQL Server中创建了非对称密钥，还没有非常简单的方法在其他用户数据库中重用相同的密钥。

非对称密钥对于数据加密来说是高安全选项，但在使用它们时也需要更多的SQL Server资源。在下面一组技巧中，我将演示如何创建、管理和使用非对称密钥进行加密。

19.3.1 创建非对称密钥

在这个技巧中，我将演示创建非对称密钥，在加密和解密数据时将用到它。创建非对称密钥的部分语法如下：

```
CREATE ASYMMETRIC KEY Asym_Key_Name
[ FROM PROVIDER Provider_Name ]
   [ AUTHORIZATION database_principal_name ]
   {FROM <Asym_Key_Source>|
      WITH ALGORITHM = <key_option>
   [ ENCRYPTION BY PASSWORD = 'password' ]
```

表19-2描述了这个命令的参数。

表19-2 CREATE ASYMMETRIC KEY的参数

参 数	描 述
Asym_Key_Name	非对称密钥的名称
Provider_Name	可扩展密钥管理的提供程序名称

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

参　数	描　述
database_principal_name	非对称密钥的拥有者
Asym_Key_Source	允许定义外部密钥源的选项（文件、程序集、EKM提供程序）
key_option	当生成新密钥时使用的算法选项，允许从安全类型（RSA_512、RSA_1024、RSA_2048）中选择。Key_option还允许从EKM提供程序指定密钥名称，并在EKM设备上创建或打开密钥
Password	用来加密私钥的密码。不使用它时，自动使用数据库主密钥对私钥进行加密

在这个示例中，在数据库BookStore创建新的非对称密钥：

```
USE BookStore
GO

CREATE ASYMMETRIC KEY asymBookSellerKey
WITH ALGORITHM = RSA_512
ENCRYPTION BY PASSWORD = 'EEB0B4DD!!!'
```

解析

这个示例演示了在数据库BookStore中创建非对称密钥。代码的第一行指定了新密钥的名称：

```
CREATE ASYMMETRIC KEY asymBookSellerKey
```

代码的第二行指定了加密安全类型：

```
WITH ALGORITHM = RSA_512
```

代码的第三行指定了用来加密非对称密钥的密码：

```
ENCRYPTION BY PASSWORD = 'EEB0B4DD!!!'
```

19.3.2　查看当前数据库中的非对称密钥

你可以通过查询系统目录视图sys.asymmetric_keys来查看当前数据库中的所有非对称密钥。例如：

```
SELECT name, algorithm_desc, pvt_key_encryption_type_desc
FROM sys.asymmetric_keys
```

这个查询返回：

```
name                 algorithm_desc      pvt_key_encryption_type_desc
asymBookSellerKey    RSA_512             ENCRYPTED_BY_PASSWORD
```

556

解析

使用系统目录视图sys.asymmetric_keys查看在当前数据库中的非对称密钥。代码的第一行指定了名称、安全类型以及私钥加密的方式。

```
SELECT name, algorithm_desc, pvt_key_encryption_type_desc
```

代码的第二行在FROM子句中指定了系统目录视图：

```
FROM sys.asymmetric_keys
```

19.3.3　修改非对称密钥的私钥密码

你也可以使用带有ENCRYPTION BY PASSWORD和DECRYPTION BY PASSWORD选项的ALTER ASYMMETRIC KEY

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

命令来修改私钥的密码。

这个技巧的示例演示了对非对称密钥使用新密码：

```
ALTER ASYMMETRIC KEY asymBookSellerKey
WITH PRIVATE KEY
(ENCRYPTION BY PASSWORD = 'newpasswordE4D352F280E0',
DECRYPTION BY PASSWORD = 'EEB0B4DD!!!')
```

解析

在这个技巧中，我们使用了ALTER ASYMMETRIC KEY命令修改私钥密码。代码的第一行指定了非对称密钥的名称：

```
ALTER ASYMMETRIC KEY asymBookSellerKey
```

在ENCRYPTION BY PASSWORD参数中指定了新密码：

```
WITH PRIVATE KEY
(ENCRYPTION BY PASSWORD = 'newpasswordE4D352F280E0',
```

在DECRYPTION BY PASSWORD参数中指定了旧私钥的密码：

```
DECRYPTION BY PASSWORD = 'EEB0B4DD!!!')
```


19.3.4 使用非对称密钥对数据进行加密和解密

因为同时需要公钥和私钥对，在维护保密数据时使用非对称密钥来加密数据是非常安全的方式。

警告 但是当用在大数据集时，使用非对称密钥进行加密相对于对称密钥的加密来说比较耗费资源，对称密钥使用一个密钥来加密和解密数据，速度更快。

授予通过非对称密钥对数据加密不再推荐使用，不过你仍然可以把它当作一个选择。一旦将非对称密钥添加到数据库，就可以使用它来加密和解密数据。使用函数EncryptByAsymKey用来加密数据。

语法如下：

```
EncryptByAsymKey ( Asym_Key_ID , { 'plaintext' | @plaintext } )
```

557

表19-3描述了这个命令的参数。

表19-3 EncryptByAsymKey的参数

参 数	描 述
Asym_Key_ID	用来加密数据的非对称密钥的ID。可以使用函数AsymKey_ID来返回非对称密钥的ID
' plaintext ' \| @plaintext	要加密的、尚未加密的文本（从字符串或局部变量）

在通过证书加密数据时，函数EncryptByAsymKey返回varbinary类型的加密数据。

使用函数DecryptByAsymKey来解密使用非对称密钥加密的数据。

语法如下：

```
DecryptByAsymKey ( Asym_Key_ID ,
                   { ' ciphertext ' | @ciphertext }
                   [ , ' Asym_Key_Password ' ] )
```

表19-4描述了这个命令的参数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表19-4　DecryptByAsymKey的参数

参　数	描　述
Asym_Key_ID	用来解密数据的非对称密钥的ID。可以使用系统函数Asym_Key_ID返回非对称密钥的ID
' ciphertext ' \| @ciphertext	要解密的已加密文本
'Asym_Key_Password'	非对称密钥的私钥的密码（创建非对称密钥时使用的密码）

在这个示例中，创建包含特定书商的银行代码信息：

```
Use BookStore
GO

CREATE TABLE dbo.BookSellerBankRouting
(BookSellerID int NOT NULL PRIMARY KEY,
 BankRoutingNBR varbinary(300) NOT NULL)
```

下一步，在已经准备加密的银行代码号码上，使用EncryptByAsymKey函数向表中插入新行：

```
INSERT dbo.BookSellerBankRouting
(BookSellerID, BankRoutingNBR)
VALUES (22,
EncryptByAsymKey(AsymKey_ID('asymBookSellerKey'),
            '137492837583249ABR'))
```

在新插入的行中查询BankRoutingNBR的值返回密文：

```
SELECT CAST(BankRoutingNBR as varchar(100)) BankRoutingNBR
FROM dbo.BookSellerBankRouting
WHERE BookSellerID = 22
```

558

这个查询返回：

```
BankRoutingNBR
m(Ì_'dc`Ó«·"ÆöÖï2ö]OE¡lå80'a8___.§6øovP¤îÎwñ@lÈ__µq-@'cda_?Lÿ<_3p'85íàj_{
```

下一步，使用函数DecryptByAsymKey来解密列BankRoutingNBR的值。我还使用CAST将varbinary数据转换为varchar数据：

```
SELECT CAST(DecryptByAsymKey
      ( AsymKey_ID('asymBookSellerKey'),
                  BankRoutingNBR,
            N'newpasswordE4D352F280E0') as varchar(100)) BankRoutingNBR
FROM dbo.BookSellerBankRouting
WHERE BookSellerID = 22
```

这个查询返回：

```
BankRoutingNBR
137492837583249ABR
```

解析

在这个技巧中，首先创建存储加密银行代码号码的表。为了存储加密的数据，将BankRoutingNBR列指定为了varbinary数据类型。然后在表上执行INSERT操作。指定了两个要插入的列：

```
INSERT dbo.BookSellerBankRouting
(BookSellerID, BankRoutingNBR)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

将BookSellerID的值设置为22：

```
VALUES (22,
```

使用函数EncryptByAsymKey填充BankRoutingNBR列：

```
EncryptByAsymKey(
```

这个函数的第一个参数是非对称密钥的系统ID，使用AsymKey_ID函数将密钥名称转换为密钥ID整数值：

```
AsymKey_ID('asymBookSellerKey'),
```

第二个参数包含了要加密的银行代码号码：

```
'137492837583249ABR'))
```

然后以密文形式将数据存储在表中。使用函数DescryptByAsymKey来解密数据。为了将varbinary值转换为varchar值，使用函数CAST：

```
SELECT CAST(
```

函数DecryptByAsymKey的第一个参数是非对称密钥的系统ID，再一次使用AsymKey_ID将非对称密钥的名称转换为ID：

```
DecryptByAsymKey
      ( AsymKey_ID('asymBookSellerKey'),
```

第二个参数是表dbo.BookSellerBankRouting的BankRoutingNBR列：

```
BankRoutingNBR,
```

第三个参数是非对称密钥的私钥的密码：

```
N'newpasswordE4D352F280E0')
```

然后将数据类型转换为varchar(100)：

```
as varchar(100)) BankRoutingNBR
FROM dbo.BookSellerBankRouting
WHERE BookSellerID = 22
```

19.3.5 删除非对称密钥

使用DROP ASYMMETRIC KEY命令删除非对称密钥。这个命令只使用了一个参数——非对称密钥的名称。

例如：

```
DROP ASYMMETRIC KEY asymBookSellerKey
```

解析

这个示例演示了删除非对称密钥的简单方法。要牢记，如果非对称密钥用来加密其他密钥或映射到登录名，那么它是不可以删除的。如果你用它直接加密数据（不推荐），你将删除解密的方法。

19.4 对称密钥加密

证书（本章稍后会介绍）和非对称密钥使用数据库级内部公钥加密数据，并且使用数据库级内部私钥解密数据。对称密钥是相对简单的，它们包含一个同时用来加密和解密的密钥。因此，使用对称密钥加密数据更快，并且用在大数据集时更加合适。尽管对加密的复杂度进行了权衡，对称密钥仍然

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

是在数据库中加密保密数据时好的选择。

在下面一组技巧中，我将演示如何创建、管理并且使用对称密钥进行加密。

19.4.1　创建对称密钥

对大数据集进行加密时，对称密钥是消耗资源更少的方式。不像证书或非对称密钥，对称密钥只使用一个内部密钥对数据加密和解密。对称密钥的特性是，在数据库会话中使用它对数据进行加密或
560 解密之前必须先打开。

使用CREATE SYMMETRIC KEY命令来创建对称密钥。部分语法如下：

```
CREATE SYMMETRIC KEY key_name [ AUTHORIZATION owner_name ]
    [ FROM PROVIDER Provider_Name ]
    WITH <key_options> [ , ... n ] |
    ENCRYPTION BY <encrypting_mechanism> [ , ... n ]
```

表19-5描述了这个命令的参数。

表19-5　CREATE SYMMETRIC KEY的参数

参　数	描　述
key_name	新对称密钥的名称。如果加上#符号前缀，可为当前会话及用户创建临时的密钥
owner_name	指定拥有密钥的数据库用户
Provider_Name	定义可扩展密钥管理提供程序的名称
Key_options	指定用于定义密钥源、算法（DES \|TRIPLE_DES \| TRIPLE_DES_3KEY \| RC2 \| RC4 \| DESX \| AES_128 \|AES_192 \| AES_256）和可选标识短语（用于生成用临时密钥加密的标签数据的GUID的基于字符的短语）的选项
Encrypting_mechanism	定义如何保护对称密钥（证书、密码、非对称密钥或其他对称密钥）

在这个技巧中，创建使用既有的数据库非对称密钥加密的新对称密钥：

```
USE BookStore
GO

-- 创建用于加密对称密钥的非对称密钥
CREATE ASYMMETRIC KEY asymBookSellerKey
WITH ALGORITHM = RSA_512
ENCRYPTION BY PASSWORD = 'EEB0B4DD!!!'

-- 创建对称密钥
CREATE SYMMETRIC KEY sym_BookStore
WITH ALGORITHM = TRIPLE_DES
ENCRYPTION BY ASYMMETRIC KEY asymBookSellerKey
```

解析

在这个技巧中演示了对称密钥的创建过程，这个对称密钥将用来加密数据。必须使用证书、密码、非对称密钥或其他对称密钥对它加密。在这个例子中，我使用了非对称密钥进行加密。

创建非对称密钥后，使用CREATE SYMMETRIC KEY命令。代码的第一行指定了对称密钥的名称：

```
CREATE SYMMETRIC KEY sym_BookStore
```

代码的第二行指定了用来创建加密密钥的加密算法：

```
WITH ALGORITHM = TRIPLE_DES
```

代码的最后一行在当前数据库中定义了用来加密对称密钥的非对称密钥：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ENCRYPTION BY ASYMMETRIC KEY asymBookSellerKey
```

561

19.4.2 查看当前数据库中的对称密钥

你可以通过查询系统目录视图sys.symmetric_keys来查看当前数据库中的对称密钥：

```
SELECT name, algorithm_desc
FROM sys.symmetric_keys
```

这个查询返回：

```
name                          algorithm_desc
##MS_DatabaseMasterKey##      TRIPLE_DES
sym_BookStore                 TRIPLE_DES
```

19

解析

查询系统目录视图sys.symmetric_keys返回当前数据库中对称密钥的名称和加密算法。注意查询sys.symmetric_keys也将返回数据库主密钥的行。

19.4.3 改变对称密钥加密方式

在这个技巧中，我将演示如何使用ALTER SYMMETRIC KEY命令修改对称密钥的加密方式。但在执行这个操作之前，必须先使用OPEN SYMMETRIC KEY命令打开它。

OPEN SYMMETRIC KEY命令的语法如下：

```
OPEN SYMMETRIC KEY Key_name DECRYPTION BY < decryption_mechanism >
```

打开密钥的解密机制取决于密钥最初是使用何种方式进行加密的。例如，使用加密密钥的私有密钥密码来打开如下的对称密钥：

```
OPEN SYMMETRIC KEY sym_BookStore
DECRYPTION BY ASYMMETRIC KEY asymBookSellerKey
WITH PASSWORD = 'EEB0B4DD!!!'
```

打开密钥之后，就可以使用密码的加密来取代密钥（先增加密码加密，然后删除非对称密钥加密）：

```
ALTER SYMMETRIC KEY sym_BookStore
ADD ENCRYPTION BY PASSWORD = 'hushhush!123'

ALTER SYMMETRIC KEY sym_BookStore
DROP ENCRYPTION BY ASYMMETRIC KEY asymBookSellerKey
```

完成这个操作之后，使用CLOSE SYMMETRIC KEY命令关闭在数据库会话中使用的密钥：

```
CLOSE SYMMETRIC KEY sym_BookStore
```

解析

这个示例演示了修改对称密钥加密的方式，在这个示例中用密码取代了使用非对称密钥进行加密。

562

首先，为了修改，使用OPEN SYMMETRIC KEY命令打开密钥，代码的第一行指定了对称密钥的名称：

```
OPEN SYMMETRIC KEY sym_BookStore
```

代码的第二行指定了用来加密对称密钥的非对称密钥的名称：

```
DECRYPTION BY ASYMMETRIC KEY asymBookSellerKey
```

代码的第三行指定了非对称密钥的私钥密码：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
WITH PASSWORD = 'EEB0B4DD!!!'
```

打开它之后，将密钥改为使用密码的加密方式。代码的第一行指定了要修改的对称密钥：

```
ALTER SYMMETRIC KEY sym_BookStore
```

代码的第二行指定了使用密码来加密对称密钥：

```
ADD ENCRYPTION BY PASSWORD = 'hushhush!123'
```

在这之后，使用ALTER SYMMETRIC KEY命令删除非对称密钥加密。第一行代码指定了要修改的对称密钥：

```
ALTER SYMMETRIC KEY sym_BookStore
```

代码的第二行指定了要删除的非对称密钥加密方式：

```
DROP ENCRYPTION BY ASYMMETRIC KEY asymBookSellerKey
```

在完成这个操作之后，使用CLOSE SYMMETRIC KEY命令关闭用在数据库会话中的密钥：

```
CLOSE SYMMETRIC KEY sym_BookStore
```

19.4.4　使用对称密钥进行加密和解密

为了使用对称密钥对数据进行加密，必须首先打开对称密钥，然后使用函数EncryptByKey。这个函数的语法如下：

```
EncryptByKey( key_GUID , { ' cleartext ' | @cleartext }
[ , { add_authenticator | @add_authenticator }
, { authenticator | @authenticator } ])
```

表19-6描述了这个命令的参数。

表19-6　EncryptByKey的参数

参　数	描　述
key_GUID	对称密钥的全局唯--标识符（GUID），它可以通过使用系统函数Key_GUID派生出来
'cleartext' \| @cleartext	要加密的文本
add_authenticator \| @add_authenticator	决定是否将验证器与明文一起加密的布尔值（1或0）。被加密的数据今后可以通过使用其他绑定值进行加密——例如表的主键。如果验证器被修改（或损坏）了，加密的数据将不能被解密
authenticator \| @authenticator	用于派生验证器的数据列。例如，你可以将加密的数据与表的主键绑定在一起

563

在这个示例中，创建用来保存客户密码提示的表。表中密码提示的答案将被加密：

```
USE BookStore
GO

CREATE TABLE dbo.PasswordHint
(CustomerID int NOT NULL PRIMARY KEY,
 PasswordHintQuestion varchar(300) NOT NULL,
 PasswordHintAnswer varbinary(200) NOT NULL)
GO
```

下一步，插入新行到使用对称密钥加密PasswordHintAnswer列的dbo.PasswordHint表中：

```
OPEN SYMMETRIC KEY sym_BookStore
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DECRYPTION BY PASSWORD = 'hushhush!123'

INSERT dbo.PasswordHint
(CustomerID, PasswordHintQuestion, PasswordHintAnswer)
VALUES
(23, 'What is the name of the hospital you were born in?',
EncryptByKey(Key_GUID('sym_BookStore '), 'Mount Marie'))

CLOSE SYMMETRIC KEY sym_BookStore
```

使用DecryptByKey命令来解密用对称密钥加密的数据。注意，不像EncryptByKey命令，DecryptByKey不用对称密钥GUID。因此为了解密数据，必须打开正确的对称密钥会话：

```
OPEN SYMMETRIC KEY sym_BookStore
DECRYPTION BY PASSWORD = 'hushhush!123'

SELECT CAST(DecryptByKey(PasswordHintAnswer) as varchar(200)) PasswordHintAnswer
FROM dbo.PasswordHint
WHERE CustomerID = 23

CLOSE SYMMETRIC KEY sym_BookStore
```

这个查询返回：

```
PasswordHintAnswer
Mount Marie
```

如果你尝试不先打开用来加密数据的对称密钥就查询前面查询中的值，则将返回NULL值：

```
PasswordHintAnswer
NULL
```

像前面EncryptByKey的语法显示的一样，你也可以为文本数据的加密方式增加其他验证器列值。这个额外的信息有助于进一步地让从有意义的值加密过来的文本变得晦涩，这个有意义的值可能会从加密的文本和表中其他非加密的列中派生出来。

在下一个示例中，我使用dbo.PasswordHint表的主键值。首先创建一个新表，并向其中添加一个未加密的行：

```
CREATE TABLE dbo.BookSellerLogins
(LoginID int NOT NULL PRIMARY KEY,
 Password varbinary(256) NOT NULL)
GO

INSERT dbo.BookSellerLogins
(LoginID, Password)
VALUES(22, CAST('myeasypassword' as varbinary))
```

下一步，打开对称密钥，并且使用对称密钥和行的LoginID列在UPDATE语句中对password列的值进行加密：

```
OPEN SYMMETRIC KEY sym_BookStore
DECRYPTION BY PASSWORD = 'hushhush!123'

UPDATE dbo.BookSellerLogins
SET Password =
    EncryptByKey(Key_GUID('sym_BookStore'),
              Password,
              1,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            CAST(LoginID as varbinary))

CLOSE SYMMETRIC KEY sym_BookStore
```

现在，为了解密这个更新行的值，在函数的调用中DecryptByKey也必须包含验证器列：

```
OPEN SYMMETRIC KEY sym_BookStore
DECRYPTION BY PASSWORD = 'hushhush!123'

SELECT LoginID,
CAST(DecryptByKey(Password, 1,
CAST(LoginID as varbinary)) as varchar(30)) Password
FROM dbo.BookSellerLogins

CLOSE SYMMETRIC KEY sym_BookStore
```

这个查询返回：

```
LoginID         Password
22              myeasypassword
```

565

解析

在这个技巧中，我演示了如何使用EncryptByKey函数加密数据和使用DecryptByKey函数解密数据。在使用函数之前，必须首先打开对称密钥。OPEN SYMMETRIC KEY命令的第一行引用了对称密钥的名称：

```
OPEN SYMMETRIC KEY sym_BookStore
```

第二行包含了用来访问要使用的对称密钥的密码：

```
DECRYPTION BY PASSWORD = 'hushhush!123'
```

然后插入使用EncryptByKey来加密值的新行。第一个参数使用Key_GUID函数返回使用的对称密钥的系统ID。第二个参数是使用对称密钥加密的文本：

```
...
EncryptByKey(Key_GUID('sym_BookStore '), 'Mount Marie'))
```

在完成加密过程后关闭密钥，参数引用了对称密钥：

```
CLOSE SYMMETRIC KEY sym_BookStore
```

再次打开对称密钥来解密数据：

```
OPEN SYMMETRIC KEY sym_BookStore
DECRYPTION BY PASSWORD = 'hushhush!123'
```

使用DecryptByKey函数，将加密数据存储所在的表列作为参数：

```
SELECT CAST(DecryptByKey(PasswordHintAnswer) as varchar(200)) PasswordHintAnswer
FROM dbo.PasswordHint
WHERE CustomerID = 23
```

在返回解密的数据之后，关闭对称密钥：

```
CLOSE SYMMETRIC KEY sym_BookStore
```

也演示了使用验证器对数据进行加密。在这个示例中，第三个参数是指示是否使用（1表示True）验证器值的标志，随后是验证器列（LoginID）：

```
EncryptByKey(Key_GUID('sym_BookStore'),
            Password,
            1,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CAST(LoginID as varbinary)
```

在数据进行加密之前，LoginID被转换为varbinary。使用验证器更进一步增强了加密数据的完全性。然而，如果特定行的相关验证器LoginID值被改变，那么加密的数据将不可再用修改的LoginID值来进行解密。

在示例中，打开对称密钥，并且使用函数DecryptByKey，包括在第一个参数中加密的列、第二个参数为验证器标志，以及第三个参数中的验证器列（被转换为varbinary数据类型）：

```
SELECT LoginID,
CAST(DecryptByKey(Password, 1,
CAST(LoginID as varbinary)) as varchar(30)) Password
FROM dbo.BookSellerLogins
```

566

在返回解密的数据之后，关闭对称密钥。

19.4.5 删除对称密钥

你可以使用DROP SYMMETRIC KEY命令从数据库中删除对称密钥，使用的参数只有一个，即对称密钥的名称。

例如：

```
DROP SYMMETRIC KEY sym_BookStore
```

解析

在这个技巧中，我演示了从数据库中删除对称密钥。它只有一个参数——对称密钥的名称。要牢记，如果打开了密钥，DROP命令将引发错误并执行失败。

19.5 证书加密

证书可以用来在数据库中加密或解密数据。证书包含密钥对、关于证书拥有者的信息以及证书可用的开始和结束过期日期。证书同时包含公钥和私钥。正如你稍后在技巧中看到的，证书的公钥用来加密数据，私钥用来解密数据。SQL Server可以生成它自己的证书，也可以从外部文件或程序集载入。因为可以备份然后从文件中载入它们，证书比非对称密钥更易于移植，而非对称密钥不能这样。这意味着在多用户数据库中可以方便地重用同一个证书。一旦创建了证书，就可以对数据库数据使用证书加密或解密函数。

提示 证书和非对称密钥数据库对象都提供了一种非常安全的加密数据的方式。但是，这个强壮的加密方式也带来了较大的性能消耗。使用证书或非对称密钥加密非常大的数据集可能会对运行环境产生很大的开销。使用对称密钥相对开销较低，我们之前已经讨论过它了。

在下面一组技巧中，我将演示如何创建、管理和使用基于证书的加密。

19.5.1 创建数据库证书

使用CREATE CERTIFICATE命令创建新数据库证书。在数据库中创建新证书的部分语法如下：

```
CREATE CERTIFICATE certificate_name [ AUTHORIZATION user_name ]
    { FROM <existing_keys> | <generate_new_keys> }
```

表19-7描述了这个命令的参数。

567

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表19-7　CREATE CERTIFICATE的参数

参　数	描　述
certificate_name	新数据库证书的名称
user_name	拥有证书的数据库用户
existing_keys	在既有的程序集或证书文件（和私有密钥文件）中创建证书的参数
generate_new_keys	用于创建新证书，包含密码、证书主题、开始时间、过期时间和私有密钥加密选项

在这个示例中，在数据库BookStore中创建新证书：

```
USE BookStore
GO

CREATE CERTIFICATE cert_BookStore
ENCRYPTION BY PASSWORD = 'AA5FA6AC!!!'
WITH SUBJECT = 'BookStore Database Encryption Certificate',
START_DATE = '2/20/2008', EXPIRY_DATE = '10/20/2009'
```

解析

在这个技巧中我创建了将用来加密或解密数据的新证书。代码的第一行包含了新证书的名称：

```
CREATE CERTIFICATE cert_BookStore
```

第二行包含了用来加密证书的密码：

```
ENCRYPTION BY PASSWORD = 'AA5FA6AC!!!'
```

第三行指定了证书的主题，随后是证书的开始和过期日期。

```
WITH SUBJECT = 'BookStore Database Encryption Certificate',
START_DATE = '2/20/2008', EXPIRY_DATE = '10/20/2009'
```

19.5.2　查看数据库中的证书

一旦在数据库中创建了证书，就可以通过查询系统目录视图sys.certificates来查看这些证书：

```
SELECT name, pvt_key_encryption_type_desc, issuer_name
FROM sys.certificates
```

这个查询返回：

```
name            pvt_key_encryption_type_desc    issuer_name
cert_BookStore  ENCRYPTED_BY_PASSWORD           BookStore ...
```

568

解析

查询系统目录视图sys.certificates来查看当前数据库中的证书。name列返回了证书的名称。结果集中的pvt_key_encryption_type_desc列描述了证书中的私钥是如何加密的。issuer_name返回了证书的主题。

19.5.3　备份和还原证书

在创建了证书之后，也可以使用BACKUP CERTIFICATE命令将它备份到文件，为了安全地保存它或是在其他数据库中还原它。

语法如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
BACKUP CERTIFICATE certname TO FILE = 'path_to_file'
    [ WITH PRIVATE KEY
        ( FILE = 'path_to_private_key_file' ,
          ENCRYPTION BY PASSWORD = 'encryption_password'
[ , DECRYPTION BY PASSWORD = 'decryption_password' ]   ) ]
```

表19-8描述了这个命令的参数。

表19-8 BACKUP CERTIFICATE的参数

参 数	描 述
path_to_file	证书备份写入的文件名和路径
path_to_private_key_file	指定私钥文件的路径和文件名
encryption_password	创建证书时使用的私钥的密码
decryption_password	在备份之前用来解密密钥的私钥密码

这个示例演示了备份证书（假定我在SQL Server实例上有对应的路径C:\Apress\Recipes\Certificates）：

```
BACKUP CERTIFICATE cert_BookStore
TO FILE = 'C:\Apress\Recipes\Certificates\certBookStore.BAK'
WITH PRIVATE KEY ( FILE = 'C:\Apress\Recipes\Certificates\certBookStorePK.BAK' ,
ENCRYPTION BY PASSWORD = '3439F6A!!!',
DECRYPTION BY PASSWORD = 'AA5FA6AC!!!' )
```

这个备份创建了两个文件，一个是包含公钥的证书（用来加密数据的），另一个是包含保护密码的私钥的证书（用来解密数据的）。

在备份之后，可以在其他数据库中使用这个证书，或使用DROP CERTIFICATE命令删除既有的证书（使用证书的名称作为它的参数）并从备份中还原它，如下所示：

```
DROP CERTIFICATE cert_BookStore
GO

CREATE CERTIFICATE cert_BookStore
FROM FILE = 'C:\Apress\Recipes\Certificates\certBookStore.BAK'
WITH PRIVATE KEY (FILE = 'C:\Apress\Recipes\Certificates\certBookStorePK.BAK',
              DECRYPTION BY PASSWORD = '3439F6A!!!',
              ENCRYPTION BY PASSWORD = 'AA5FA6AC!!!')
```

解析

这个技巧演示了使用BACKUP CERTIFICATE命令将证书备份到外部文件，使用DROP CERTIFICATE命令删除它，然后使用CREATE CERTIFICATE命令从文件中重新创建它。

浏览代码，备份操作的第一行引用了证书的名称：

```
BACKUP CERTIFICATE cert_BookStore
```

TO FILE子句包括了证书的公钥将备份到的文件名：

```
TO FILE = 'C:\Apress\Recipes\Certificates\certBookStore.BAK'
```

WITH PRIVATE KEY子句指定了私钥备份输出到的文件名以及加密（创建证书时使用的私钥密码）密码和解密（在备份之前用来解密密钥的私钥密码）密码：

```
WITH PRIVATE KEY ( FILE = 'C:\Apress\Recipes\Certificates\certBookStorePK.BAK' ,
ENCRYPTION BY PASSWORD = '3439F6A!!!',
DECRYPTION BY PASSWORD = 'AA5FA6AC!!!' )
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在使用DROP CERTIFICATE命令删除既有证书之后，然后从备份文件中重建证书。CREATE CERTIFICATE命令的第一行引用了证书的名称：

```
CREATE CERTIFICATE cert_BookStore
```

FROM FILE子句指定了公钥备份文件的位置：

```
FROM FILE =
'C:\Apress\Recipes\Certificates\certBookStore.BAK'
```

WITH PRIVATE KEY子句指定了私钥文件的位置，随后是解密和加密密码：

```
WITH PRIVATE KEY (FILE = 'C:\Apress\Recipes\Certificates\certBookStorePK.BAK',
              DECRYPTION BY PASSWORD = '3439F6A!!!',
              ENCRYPTION BY PASSWORD = 'AA5FA6AC!!!')
```

19.5.4 管理证书的私钥

可以通过使用ALTER CERTIFICATE 命令为证书增加或删除私钥。这个命令允许删除私钥（默认通过数据库主密钥进行加密）、增加私钥或修改私钥的密码。

随后的示例从证书中删除私钥：

```
ALTER CERTIFICATE cert_BookStore
REMOVE PRIVATE KEY
```

和CREATE CERTIFICATE命令一样，也可以使用ALTER CERTIFICATE命令从备份文件为既有证书重新增加私钥：

```
ALTER CERTIFICATE cert_BookStore
WITH PRIVATE KEY
(FILE = 'C:\Apress\Recipes\Certificates\certBookStorePK.BAK',
       DECRYPTION BY PASSWORD = '3439F6A!!!',
            ENCRYPTION BY PASSWORD = 'AA5FA6AC!!!')
```

570

也可以使用ALTER CERTIFICATE命令修改既有私钥的密码：

```
ALTER CERTIFICATE cert_BookStore
WITH PRIVATE KEY (DECRYPTION BY PASSWORD = 'AA5FA6AC!!!',
ENCRYPTION BY PASSWORD = 'mynewpassword!!!Efsj')
```

DECRYPTION BY PASSWORD是旧的私钥密码，ENCRYPTION BY PASSWORD是新的私钥密码。

解析

这个技巧演示了如何修改证书的加密方式。使用ALTER CERTIFICATE和REMOVE PRIVATE KEY命令从证书中删除私钥：

```
ALTER CERTIFICATE cert_BookStore
REMOVE PRIVATE KEY
```

我们也使用ALTER CERTIFICATE命令将它再次添加回去。第一行引用了证书的名称：

```
ALTER CERTIFICATE cert_BookStore
```

WITH PRIVATE KEY子句指定了私钥文件的位置，以及解密密码和加密密码：

```
WITH PRIVATE KEY
(FILE = 'C:\Apress\Recipes\Certificates\certBookStorePK.BAK',
        DECRYPTION BY PASSWORD = '3439F6A!!!',
             ENCRYPTION BY PASSWORD = 'AA5FA6AC!!!')
```

最后，我修改了证书的私钥密码。第一行引用了证书的名称：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ALTER CERTIFICATE cert_BookStore
```

WITH PRIVATE KEY子句指定了解密密码以及新的加密密码：

```
WITH PRIVATE KEY (DECRYPTION BY PASSWORD = 'AA5FA6AC!!!',
ENCRYPTION BY PASSWORD = 'mynewpassword!!!Efsj')
```

19.5.5 使用证书加密和解密

一旦在数据库中拥有了证书，就可以使用系统函数EncryptByCert用证书的公钥来加密数据。加密允许保护敏感表数据。如果没有关联的私钥，数据将不可读取。

系统函数EncryptByCert的语法如下：

```
EncryptByCert ( certificate_ID , { ' cleartext ' | @cleartext } )
```

表19-9描述了这个命令的参数。

表19-9 EncryptByCert的参数

参 数	描 述
certificate_ID	用来加密数据的证书ID
'cleartext' \| @cleartext	要加密的尚未加密的文本

可以使用函数Cert_ID来检索特定数据库证书的证书ID，该函数将证书名称作为唯一的参数：

```
Cert_ID ( ' cert_name ' )
```

使用函数DecryptByCert来解密通过证书加密的数据。这个函数使用证书的内部私钥来解密数据（当创建证书时，私钥需要定义私钥密码）：

```
DecryptByCert ( certificate_ID ,
              { ' ciphertext ' | @ciphertext }
              [ , { ' cert_password ' | @cert_password } ] )
```

表19-10描述了这个命令的参数。

表19-10 DecryptByCert的参数

参 数	描 述
certificate_ID	用来解密数据的证书ID
'ciphertext' \| @ciphertext	要解密的已加密的文本
'cert_password' \| @cert_password	用来解密数据的证书的私钥密码

在这个示例中，在表PasswordHintAnswer中使用INSERT，数据是经过证书的公钥加密的：

```
USE BookStore
GO

INSERT dbo.PasswordHint
(CustomerID, PasswordHintQuestion, PasswordHintAnswer)
VALUES
(1, 'What is the name of the hospital you were born in?',
   EncryptByCert(Cert_ID('cert_BookStore'), 'Hickman Hospital'))
```

下一个查询显示了新插入的行：

```
SELECT CAST(PasswordHintAnswer as varchar(200)) PasswordHintAnswer
FROM dbo.PasswordHint
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
WHERE CustomerID = 1
```

这个查询返回难以理解的密文，而不是原始的文本值：

```
PasswordHintAnswer
o<_*_1/2bYy-X-_Î`'5BuÃ*n«ßR_.´jõÑ†£sÙ_"ùüÔ_ÄÆ7(c)±w__Åa_3U_'c9_>¯
```

下一个示例演示了查询PasswordHintAnswer列，这次使用证书的私钥来查看解密的结果：

```
SELECT CAST(DecryptByCert(
                Cert_ID('cert_BookStore'),
                PasswordHintAnswer,
                N'mynewpassword!!!Efsj')
                   as varchar(200)) PasswordHintAnswer
FROM dbo.PasswordHint
WHERE CustomerID = 1
```

572 这个查询返回：

```
PasswordHintAnswer
Hickman Hospital
```

解析

在这个技巧的示例中，使用varbinary数据类型列来保存加密的信息，并以此为基础创建了表。因为函数EncryptByCert函数返回varbinary类型的加密数据，所以选择了这个数据类型。函数EncryptByCert的第一个参数为证书ID，随后是要加密的文本：

```
EncryptByCert(Cert_ID('cert_BookStore'), 'Hickman Hospital')
```

要加密的文本可以为nvarchar、varchar、char或nchar数据类型。但是，数据实际被存储为varbinary类型。如果你尝试将varbinary数据转换为原始的文本数据类型，不使用函数DecryptByCert和适当的证书及密码，那么只会返回加密的乱码。

然后使用同一证书的私钥来对加密的字符串进行解密。函数的第一个参数是证书ID，第二个参数为加密的文本，第三个参数是创建证书时使用的私钥密码。

```
DecryptByCert(Cert_ID('cert_BookStore'), PasswordHintAnswer,
 N'mynewpassword!!!Efsj') PasswordHintAnswer
```

为了显示原始的文本，函数的结果也将转换为varchar数据类型。

19.5.6　通过对称密钥自动打开和解密

在本章的前面，你看到了打开用非对称密钥加密的对称密钥的演示。这个操作有两个步骤，第一步是OPEN SYMMETRIC KEY命令，随后是实际的DecryptByKey函数调用。SQL Server也提供了能够将两个前面提到的步骤组合到一个操作中的两个额外的解密函数，它们分别是：用于使用非对称密钥加密的对称密钥的DecryptByKeyAutoAsymKey和用于使用证书加密的对称密钥的DecryptByKeyAutoCert中。

DecryptByKeyAutoAsymKey和DecryptByKeyAutoCert对它们各自的对象使用相似的语法，只是它们包括一个包含非对称密钥或证书的密码的参数。

在这个技巧中，首先创建新的非对称密钥，然后通过新的非对称密钥加密对称密钥（在这个例子中非对称密钥通过数据库主密钥进行加密，因此不需要使用密码）。相关代码如下：

```
USE BookStore
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE ASYMMETRIC KEY asymBookSell_V2
WITH ALGORITHM = RSA_512

CREATE SYMMETRIC KEY sym_BookStore_V2
WITH ALGORITHM = TRIPLE_DES
ENCRYPTION BY ASYMMETRIC KEY asymBookSell_V2
```

573

下一步，向dbo.PasswordHint表插入新行：

```
OPEN SYMMETRIC KEY sym_BookStore_V2
DECRYPTION BY ASYMMETRIC KEY asymBookSell_V2

INSERT dbo.PasswordHint
(CustomerID, PasswordHintQuestion, PasswordHintAnswer)
VALUES
(45, 'What is the name of the hospital you were born in?',
EncryptByKey(Key_GUID('sym_BookStore_V2'), 'Sister Helen'))

CLOSE SYMMETRIC KEY sym_BookStore_V2
```

现在我将演示DecryptByKeyAutoAsymKey函数，它可以避免使用两个操作来解密表值：

```
SELECT CAST(
   DecryptByKeyAutoAsymKey
   (ASYMKEY_ID('asymBookSell_V2'),
    NULL,
    PasswordHintAnswer) as varchar)
FROM dbo.PasswordHint
WHERE CustomerID = 45
```

这个查询返回：

```
Sister Helen
```

解析

在这个技巧中，我创建了新的非对称密钥，以及通过新创建的非对称密钥加密的对称密钥。在这之后，打开密钥并插入新行，加密了PasswordHintAnswer列值。为了查看解密的数据，我并没有重新打开对称密钥。而是调用了DecryptByKeyAutoAsymKey函数。调用函数的第一个参数是非对称密钥的非对称密钥ID：

```
SELECT CAST(
   DecryptByKeyAutoAsymKey
   (ASYMKEY_ID('asymBookSell_V2'),
```

第二个参数是非对称密钥的密码。由于创建的非对称密钥是隐式地用数据库主密钥进行加密的，所以我对这个参数使用了NULL值：

```
NULL,
```

第三个参数是要解密的加密列：

```
    PasswordHintAnswer) as varchar)
FROM dbo.PasswordHint
WHERE CustomerID = 45
```

整个函数的调用都包含在CAST函数中，用来以varchar数据类型显示结果。

574

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

19.6 透明数据加密

SQL Server 2008引入了透明数据加密，或称TDE，它允许你不修改应用程序的代码来完整地加密数据库文件。当用户数据库可用并且启用了TDE时，数据写入磁盘时便在页级别进行了加密。当数据页读入到内存时进行解密。假如数据文件或数据库备份被偷，如果没有原始加密它的证书数据是不能读取的。

随后的两个技巧将演示如何启用和管理透明数据加密。

19.6.1 启用透明数据加密

在这个技巧中，我将演示如何对用户定义的数据库启用透明数据加密。准备TDE的数据库，第一步是在master系统数据库中创建主密钥和证书：

```
USE master
GO

CREATE MASTER KEY ENCRYPTION
BY PASSWORD = '834BACDA-10E6-4BBC-A698-952533E54337'
GO

CREATE CERTIFICATE TDE_Server_Certificate
WITH SUBJECT = 'Server-level cert for TDE'
GO
```

现在我拥有了服务器级别的证书，我将使用它来加密要启用TDE的数据库的数据库加密密钥。DEK是实际用来加密整个数据库的加密密钥。语法相对简单，因此我继续技巧并在“解析”部分更详细地解释其中一些选项。我在希望加密的数据库的上下文中执行如下命令：

```
USE BookStore
GO

CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = TRIPLE_DES_3KEY
ENCRYPTION BY SERVER CERTIFICATE TDE_Server_Certificate
GO
```

现在我已经定义好了DEK，这个技巧的最后一步是真正地加密数据库，使用ALTER DATABASE命令和ENCRYPTION选项来设置它：

```
ALTER DATABASE BookStore
SET ENCRYPTION ON
GO
```

可以对sys.databases执行如下的查询，来验证数据库已被真正加密：

```
SELECT is_encrypted
FROM sys.databases
WHERE name = 'BookStore'
```

575

这个查询返回1（true）：

```
is_encrypted
1
```

解析

这个技巧演示了对数据库启用TDE需要的每个步骤。在这个技巧的第一步，我假设主密钥尚未在

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

master系统数据库中创建。然后我创建了主密钥，以及将用于加密DEK的证书。

然后切换数据库上下文到用户定义数据库并创建了DEK。代码的第一行是标准的——我不需要定义密钥的特定名称：

```
CREATE DATABASE ENCRYPTION KEY
```

在命令的第二行中，我定义了算法强度，可选的算法强度包括AES_128、AES_192、AES_256和TRIPLE_DES_3KEY。

```
WITH ALGORITHM = TRIPLE_DES_3KEY
```

命令的最后一行定义了加密DEK要使用哪个服务器级别的证书（存在于master数据库中）：

```
ENCRYPTION BY SERVER CERTIFICATE TDE_Server_Certificate
GO
```


在为BookStore数据库创建了DEK之后，就可以使用ALTER DATABASE命令启用TDE了。第一行定义了目标数据库：

```
ALTER DATABASE BookStore
```

第二行定义ENCRYPTION应该设为ON：

```
SET ENCRYPTION ON
GO
```

在启用后，你的数据文件“静止的”被加密。没有用于加密DEK的服务器证书，被偷的数据库是不可以被还原或正常地附加到其他SQL Server实例上，也不可以对文件本身进行破解。

如果DBA想正常地将数据库从一个SQL Server实例移动到另一个实例，他可以备份服务器级别的证书并在第二个服务器中创建它。这将允许还原TDE的备份或附加数据/日志文件。

19.6.2 管理和移除 TDE

下面的技巧将演示管理和从数据库中移除TDE的通常任务。

在第一个查询中，我将演示重建既有DEK的算法强度。原始的DEK定义为TRIPLE_DES_3KEY，在下一查询中，我将它更改为AES_128：

```
ALTER DATABASE ENCRYPTION KEY
REGENERATE WITH ALGORITHM = AES_128
```

576

查询新的目录视图sys.dm_database_encryption_keys验证这个更改：

```
SELECT DB_NAME(database_id) databasenm,
       CASE encryption_state
          WHEN 0 THEN 'No encryption'
          WHEN 1 THEN 'Unencrypted'
          WHEN 2 THEN 'Encryption in progress'
          WHEN 3 THEN 'Encrypted'
          WHEN 4 THEN 'Key change in progress'
          WHEN 5 THEN 'Decryption in progress'
       END encryption_state,
       key_algorithm,
       key_length
FROM sys.dm_database_encryption_keys
```

这个查询返回所有数据库和相关DEK状态的信息。注意，结果集中有两行。tempdb数据库包含在其中，现在也必须对其加密数据——因为对所有用户数据库的加密处理也包括对tempdb的处理：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Databasenm      encryption_state    key_algorithm    key_length
Tempdb          Encrypted           AES              256
BookStore       Encrypted           AES              128
```

除了更改DEK的算法，我也可以更改用来加密DEK的服务器级别证书（可能你会觉得服务器证书是折中的办法或是你必须定期地更改它）：

```
USE master
GO

CREATE CERTIFICATE TDE_Server_Certificate_V2
WITH SUBJECT = 'Server-level cert for TDE V2'
GO

USE BookStore
GO

ALTER DATABASE ENCRYPTION KEY
ENCRYPTION BY SERVER CERTIFICATE TDE_Server_Certificate_V2
```

执行如下命令从数据库中移除TDE：

```
ALTER DATABASE BookStore
SET ENCRYPTION OFF
GO
```

这时，我可以清空工作并删除DEK（你可能需要在移除密钥之前给SQL Server一些时间来解密数据——特别是较大的数据库）：

```
USE BookStore
GO

DROP DATABASE ENCRYPTION KEY
```

577

解析

这个技巧演示了如何管理配置了TDE和相关联的DEK的数据库。可以使用ALTER DATABASE ENCRYPTION KEY命令修改DEK算法以及加密DEK的服务器级别证书（在master数据库中）。使用ALTER DATABASE和SET ENCRYPTION OFF命令移除加密。这时，我可以删除数据库的DEK。如果这是在SQL Server实例中最后一个使用TDE的用户定义数据库，在SQL Server实例重启后系统数据库tempdb也将转换为不加密的状态。

578

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第20章 Service Broker

使用Service Broker可以创建异步的、数据驱动的消息应用程序。使用Service Broker，当消息被控制在它们自己的时间段内时，应用程序任务可以持续地运行。Service Broker允许一个数据库发送消息到其他数据库，而不需要等待响应，因此即使远程数据库不能立即处理这些消息，发送数据库也将继续它的操作。因为可以使用SQL Server 的T-SQL对象和命令来完全地管理Service Broker，所以本书会介绍Service Broker。

Service Broker为SQL Server提供消息队列。这提供了从数据库中发送异步事务性消息到队列的方法，在队列中这些消息将会被其他服务获取和处理，该服务可能运行在其他数据库或服务器上。再者，对于异步程序，发送一条消息，并且应用程序不需要等待原始消息已被接收或处理的确认信息，就可以处理其他相关的任务。一旦完成了特定任务，两个Service Broker服务之间的会话就显式地结束。

Service Broker包括一些开箱即用的特性，它处理当你试图创建自己的异步消息系统时可能经常遇到的复杂因素。例如，Service Broker消息要保证以适当的顺序（proper order）或者以它们发送的原始顺序进行接收。这些消息也只会被接收一次（调度程序保证不会重复地读取）并且可以作为同一会话的一部分发送，与任务的同一实例相关。Service Broker带来的另一个好处是保证了消息的发送。如果当尝试发送第一个消息时目标数据库（第一个消息的接收者）不可用，消息将加入到发送方的数据库（第一个消息的发送者）的队列中，当目标数据库变为可用时将会尝试发送这个消息。这些消息在数据库发生故障的情况下也是可恢复的，由于Service Broker是内建在SQL Server数据库中的，可以与数据库的其余部分一起备份。

本章将通过设置BookStore/BookDistribution Service Broker应用程序，使用存储过程自动地在Service Broker队列中处理消息，并且启用Service Broker应用程序与远程的SQL Server实例进行通信，来讲解新的Service Broker对象和命令。对于非常活跃的Service Broker应用程序，SQL Server 2008引入了使用CREATE BROKER PRIORITY命令对会话设定优先级的功能，也将在本章中演示这个内容。使用这个功能，可以指定不重要或重要会话的优先级，以确保消息合理地处理。

本章最后将介绍和演示新的事件通知功能。使用Service Broker的事件通知功能可以跟踪数据库和SQL Server实例的事件（与SQL Trace相似）——其事件通知是异步的并且对SQL Server实例整体性能影响最小（与SQL Trace不同点）。

20.1 示例场景：在线书店

在假设的在线书店中，顾客生成图书订单。顾客对BookStore数据库发出购买请求且记录在数据库中，具体是使用内建的Service Broker功能将消息发送到BookDistribution数据库。数据库BookDisribution由分离的应用程序使用，该应用程序控制仓库入库和出库交付。这两个不同的数据库

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

可以存在于相同或不同的SQL Server实例中。

继续这个假设的示例，BookStore数据库通过提交某本书的订单消息开始与其他数据库的会话。此订单被发送到BookDistribution数据库的队列中，在这里接收服务程序可以立即获取消息或延迟处理消息。因为通信被异步处理，所以BookStore数据库上的原始事务不会被挂起。例如，应用程序可以处理其他任务，比如发送订单确认消息给顾客或更新应用程序中其他相关表。当BookDistribution数据库准备好处理订单时，Service Broker允许它从队列中采集消息，提取出信息，并处理这些信息。然后BookDistribution程序可以将消息发送回BookStore数据库来确认订单已接收，然后执行其余操作将书送到顾客手中。

20.2 创建基本的 Service Broker 应用程序

在下面一组技巧中，将演示设置一个将图书订单放到BookStore数据库的应用程序。这个图书订单被异步地发送到同一SQL Server实例的BookDistribution数据库中。一旦BookDistribution数据库能够处理订单，它就将发送订单确认响应。然后这个任务就完成了，并且Service Broker服务之间的会话结束。

下面是当两个数据库都在同一SQL Server实例上时，将Service Broker应用程序放在一起的大体步骤。

(1) 定义希望应用程序执行的异步任务。Service Broker对于执行松耦合操作的应用程序来说是不错的，比如触发消息，但是要在几分钟、几小时或几天之后响应，与此同时可以让其他应用程序任务继续执行其他操作。

(2) 确定Service Broker的发起方服务和目标服务是否创建在同一SQL Server实例中，或是分成两个SQL Server实例。多实例的通信还需要创建经过证书或NT安全的身份认证，并且要创建端点、路由以及对话安全模式。本章稍后会演示这些内容。

(3) 如果没有启用，则在多方参与的数据库中使用ALTER DATABASE命令设置ENABLE_BROKER以及TRUSTWORTHY数据库选项。

(4) 为所有多方参与的数据库创建数据库主密钥（更多关于数据库主密钥的内容请看第19章）。

(5) 创建希望用来在服务之间发送的消息类型。消息类型定义了从Service Broker端点（发起方服务或目标服务）发送的消息内包含的数据类型。这些消息类型应该添加到Service Broker应用程序中两个多方参与的数据库中。

(6) 创建约定来定义可以由发起方发送的各种消息以及由目标发送的消息类型的种类。约定定义了在任务级别可以发送或接收何种消息类型。约定应该添加到两个多方参与的数据库中。

580

(7) 同时在两个多方参与的数据库中创建用于保存消息的队列。队列保存消息。你可以用SELECT语句来查询队列或用RECEIVE命令来检索队列中一条或多条消息。也可以用活动存储过程或当队列中接收消息时自动处理消息的应用程序来定义所有队列。

(8) 同时在绑定特定约定到特定队列的多方参与的数据库中创建服务。服务定义了用来绑定消息队列到一个或多个约定的端点。

在完成了这些步骤之后，就为创建新对话会话（对话会话是在服务之间交换消息的操作）以及在Service Broker服务之间进行发送/接收消息做好了准备。这一节的第一个技巧将演示如何为Service Broker活动启用SQL Server数据库。

注解 正如你在本章中看到的，一些Service Broker命令要求：如果语句不是批处理中的第一个语句，之前的语句结尾必须用分号结束符。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

20.2.1 启用数据库的 Service Broker 活动

首先，在数据库master中使用ALTER DATABASE命令，来确认多方参与的数据库中ENABLE_BROKER和TRUSTWORTHY数据库设置是否启用：

```
USE master
GO

IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'BookStore')

CREATE DATABASE BookStore
GO

IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'BookDistribution')

CREATE DATABASE BookDistribution
GO

ALTER DATABASE BookStore SET ENABLE_BROKER
GO
ALTER DATABASE BookStore SET TRUSTWORTHY ON
GO

ALTER DATABASE BookDistribution SET ENABLE_BROKER
GO
ALTER DATABASE BookDistribution SET TRUSTWORTHY ON
GO
```


解析

这个技巧使用了ALTER DATABSE命令来启用数据库中的Service Broker活动。可以使用DISABLE_BROKER数据库选项来禁用Service Broker。

20.2.2 创建加密用的数据库主密钥

Service Broker在跨数据库的会话中使用对话安全模式。为了这个安全模式能够生效，所有多方参与的数据库必须拥有数据库主密钥。

在BookStore数据库中，为了允许在两个本地数据库之间加密数据，创建了数据库主密钥：

```
USE BookStore
GO

CREATE MASTER KEY
ENCRYPTION BY PASSWORD = 'D4C86597'
GO
```

下面在BookDistribution数据库中完成同样的操作：

```
USE BookDistribution
GO

CREATE MASTER KEY
ENCRYPTION BY PASSWORD = '50255686DDC5'
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

更多关于如何在数据库中创建数据库主密钥的内容请看第19章。在这个例子中，为每个在Service Broker应用程序中参与的数据库创建了一个数据库主密钥。

20.2.3　管理消息类型

消息类型定义了从Service Broker端点（发起方或目标）发送的消息中包含的数据类型。将消息类型想象成消息模板（但不是实际的消息），定义名称、拥有者以及消息内容的类型。

使用CREATE MESSAGE TYPE命令来创建新的消息类型。它的语法如下：

```
CREATE MESSAGE TYPE message_type_name
[ AUTHORIZATION owner_name ]
[ VALIDATION =
{ NONE | EMPTY  |
WELL_FORMED_XML |
VALID_XML WITH SCHEMA COLLECTION schema_collection_name } ]
```

表20-1描述了这个命令的参数。

表20-1　CREATE MESSAGE TYPE的参数

参　数	描　述
message_type_name	这个选项定义了消息类型的名称
owner_name	这个参数指定了消息类型的数据库拥有者
NONE \|EMPTY \|WELL FORMED XML \| VALID XML WITH SCHEMA COLLECTION schema_collection_name	这些设置定义了消息有效性。当为NONE时，不执行有效性检查。当为EMPTY时，消息主体必须为NULL，当选择了WELL FORMED XML时，主体必须包含合式的XML。当选择了VALID XML WITH SCHEMA COLLECTION时，消息主体必须遵守特定的XML架构

582

继续在线书店的示例，首先创建的Service Broker对象是在数据库之间交换的两个消息类型。第一个是用于发送图书订单的消息类型：

```
Use BookStore
GO

CREATE MESSAGE TYPE [//SackConsulting/SendBookOrder]
VALIDATION = WELL_FORMED_XML
GO
```

第二个消息类型是通过目标数据库发送的，用来确认已经接收到了图书订单。两个消息类型都用格式正确的XML消息主体，它的意思是消息内容必须支持有效的XML，而不是执行在消息内容上的基于架构的有效性检查：

```
CREATE MESSAGE TYPE [//SackConsulting/BookOrderReceived]
VALIDATION = WELL_FORMED_XML
GO
```

现在将数据库上下文更改到BookDistribution数据库上，而且在BookStore数据库上创建的相同的消息类型和约定也将创建在BookDistribution数据库上。如果不创建相同的消息类型，接收数据库将不能接受进入的消息。通信结构是个双向通道，每一边都要理解在对话会话中交换的消息：

```
USE BookDistribution
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE MESSAGE TYPE [//SackConsulting/SendBookOrder]
VALIDATION = WELL_FORMED_XML
GO

CREATE MESSAGE TYPE [//SackConsulting/BookOrderReceived]
VALIDATION = WELL_FORMED_XML
GO
```

解析

在上述示例中的多方参与的数据库中创建了两个不同的消息类型。在代码第一行的CREATE MESSAGE TYPE中，在中括号中指定了名称。这是将用来发送图书订单消息的消息类型的名称：

```
CREATE MESSAGE TYPE [//SackConsulting/SendBookOrder]
```

583

消息有效性类型指定为合式的XML：

```
VALIDATION = WELL_FORMED_XML
GO
```


使用同一有效性类型创建另一消息类型，这次使用不同的消息类型名称。这是用来响应图书订单消息的消息类型：

```
CREATE MESSAGE TYPE [//SackConsulting/BookOrderReceived]
VALIDATION = WELL_FORMED_XML
GO
```

注意我没有真地定义消息的内容。实际的消息是消息类型的实例，在20.2.7节的技巧中会演示这个内容。

20.2.4 创建约定

约定定义了在任务级别可以发送或接收何种消息类型。比如，任务具体可以为“将图书订单放置到分发中心”。应用程序中的所有任务应该分别定义约定，基于会话的发起方和目标之间交换的消息的类型。约定也定义了消息类型的方向（发起方到目标，目标到发起方）。

使用CREATE CONTRACT命令来创建新的约定。部分语法如下：

```
CREATE CONTRACT contract_name
   [ AUTHORIZATION owner_name ]
(  {   message_type_name
SENT BY
{ INITIATOR | TARGET | ANY }
} [ ,...n] )
```

表20-2描述了这个命令的参数。

表20-2 CREATE CONTRACT的参数

参　数	描　述
contract_name	这个参数定义新约定的名称
owner_name	这个参数指定约定的数据库拥有者
message_type_name	这个参数定义在约定中包含的消息类型的名称
INITIATOR \| TARGET \| ANY	SENT BY选项定义了消息类型可以发送的方向。当为INITIATOR时，只有开始会话的服务可以发送特定消息类型。当为TARGET时，只有会话的目标可以发送特定消息类型。ANY允许发起方和目标都发送消息
[,...n]	在约定的定义中可以定义多个消息类型

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

继续在线书店的示例，在BookStore数据库中创建用来定义可以通过发起方（BookStore数据库）或目标（BookDistribution数据库）来发送何种消息的约定：

```
Use BookStore
GO
CREATE CONTRACT
    [//SackConsulting/BookOrderContract]
    ( [//SackConsulting/SendBookOrder]
        SENT BY INITIATOR,
    [//SackConsulting/BookOrderReceived]
        SENT BY TARGET
   )
GO
```

584

现在，将上下文切换到BookDistribution数据库，然后创建相同的约定：

```
USE BookDistribution
GO

CREATE CONTRACT
    [//SackConsulting/BookOrderContract]
    ( [//SackConsulting/SendBookOrder]
          SENT BY INITIATOR,
    [//SackConsulting/BookOrderReceived]
          SENT BY TARGET
   )
GO
```

解析

这个技巧演示了在两个BookStore数据库和BookDistribution中创建新的约定。为了会话能够成功，发起方和目标的约定定义必须相同。CREATE CONTRACT命令的第一个参数包含了约定的名称：

```
CREATE CONTRACT
   [//SackConsulting/BookOrderContract]
```

在圆括号内，指定了在前面技巧中创建的被允许的消息类型，并定义了哪个角色可以使用消息类型：

```
    ( [//SackConsulting/SendBookOrder]
          SENT BY INITIATOR,
      [//SackConsulting/BookOrderReceived]
          SENT BY TARGET
    )
GO
```

消息[//SackConsulting/SendBookOrder]从BookStore数据库(INITIATOR)发送，且[//SackConsulting/BookOrderReceived]从BookDistribution数据库（TARGET）发送。

20.2.5　创建队列

队列用来保存数据。可以通过SELECT语句查询队列或使用RECEIVE命令从队列检索一条或多条消息。

创建队列之后，可以将队列绑定到当消息到达时处理消息的存储过程（请看20.3节中的技巧）。检索程序也可以是SQL Server外部的程序（比如基于.NET的程序）——但是存储过程提供了易于实现的解决方案。

使用CREATE QUEUE命令来创建新的队列，语法如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE QUEUE <object>
   [ WITH
     [ STATUS = { ON | OFF } [ , ] ]
     [ RETENTION = { ON | OFF } [ , ] ]
     [ ACTIVATION (
         [ STATUS = { ON | OFF } , ]
           PROCEDURE_NAME = <procedure> ,
           MAX_QUEUE_READERS = max_readers ,
           EXECUTE AS { SELF | 'user_name' | OWNER }
            ) ]
]
 [ ON { filegroup | [ DEFAULT ] } ]
```

表20-3描述了这个命令的参数。

表20-3 CREATE QUEUE的参数

参 数	描 述
object	这个参数定义数据库、架构和新队列的名称
STATUS = { ON \| OFF }	当STATUS为ON时，队列是可用的。当为OFF时，不可以从队列添加或删除消息
RETENTION = { ON \| OFF }	当RETENTION为ON时，则队列发送或接收的所有消息都将保存在队列中，直到会话结束为止（允许延时查看会话中正在处理的消息）。当为OFF时（它是默认值），消息在发送或接收（以及检索）之后不会保留。保留的消息对于队列中的报告是可用的，但不可以重复对其进行发送或接收
STATUS	当为ON时，指定的存储过程将被激活，以接收消息（最大数量为在max_readers参数指定的值）。当为OFF时，不会激活队列的存储过程
PROCEDURE_NAME	这个参数表示处理队列中消息的存储过程的名称。这可以为完整限定的，即使用数据库名称、架构以及存储过程名称
MAX_QUEUE_READERS = max_readers	指定可以同时激活的队列读取存储过程的最大实例数。值为从0到32 767之间的数字
EXECUTE AS	EXECUTE AS定义用于运行存储过程的数据库用户账户 当为SELF时，它运行于当前用户的上下文中。也可以指定特定的用户名称
ON { filegroup [DEFAULT] }	像表一样，队列可以放置到特定的文件组中，如果没有显式地指定，队列会被放在DEFAULT文件组中

继续在线书店应用程序的示例，在BookStore数据库中创建队列来保存从BookDistribution数据库过来的消息。下面以启用的状态创建它：

```
Use BookStore
GO

CREATE QUEUE BookStoreQueue
WITH STATUS=ON
GO
```

CREATE QUEUE命令也可以使用将自动处理消息的程序绑定到它的激活选项。在20.3节将对此进行演示。但是在此期间，将手动控制队列中信息的交换。

下一步，在BookDistribution数据库中，为从BookStore数据库中接收到的消息创建一个新队列：

```
USE BookDistribution
GO

CREATE QUEUE BookDistributionQueue
WITH STATUS=ON
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个示例中，在两个数据库中创建了队列。在BookStore中创建的第一个队列名为BookStore-Queue：

```
CREATE QUEUE BookStoreQueue
```

代码的第二行指定了以启用的状态创建队列：

```
WITH STATUS=ON
```

BookDistribution数据库中创建了第二个队列，并使用了不同的名称BookDistributionQueue。它也是以启用的状态创建的。

20.2.6　创建服务

服务定义端点，然后使用它来将消息队列绑定到一个或多个约定上。服务使用队列和约定来定义一个或一组任务。

服务是消息的发起方和接收方强制约定的规则，并将消息路由到正确的队列。

使用CREATE SERVICE命令来创建新的服务。部分语法如下：

```
CREATE SERVICE service_name
   [ AUTHORIZATION owner_name ]
   ON QUEUE [ schema_name. ]queue_name
   [ ( contract_name [ ,...n ] ) ]
```

表20-4描述了这个命令的参数。

587

表20-4　CREATE SERVICE的参数

参　数	描　述
service_name	新服务的名称
owner_name	拥有服务的数据库用户或角色
[schema_name.]queue_name	接收消息的队列名称
contract_name [,...n]	可以发送消息到新服务的约定名称。如果没有指定，新服务可以只发起（但没有接收）消息。如果只有一个发起方，所有约定都可以用来发送消息

继续在线书店的示例，在BookStore数据库中创建将队列绑定到特定约定的服务：

```
Use BookStore
GO

CREATE SERVICE [//SackConsulting/BookOrderService]
    ON QUEUE dbo.BookStoreQueue
    ([//SackConsulting/BookOrderContract])
GO
```

现在上下文切换到BookDistribution数据库，并且创建将队列绑定到约定的服务：

```
USE BookDistribution
GO

CREATE SERVICE [//SackConsulting/BookDistributionService]
    ON QUEUE dbo.BookDistributionQueue
  ([//SackConsulting/BookOrderContract])
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个技巧中，在BookStore数据库和BookDistribution中创建了服务，使用CREATE SERVICE命令来将队列绑定到约定。

创建BookStore服务的CREATE SERVICE中的第一个参数是服务的名称：

```
CREATE SERVICE [//SackConsulting/BookOrderService]
```

代码第二行指定了约定要绑定到的队列（用来接收消息的）：

```
ON QUEUE dbo.BookStoreQueue
```

第三个参数是绑定到队列并且通过服务暴露出来的约定名称：

```
([//SackConsulting/BookOrderContract])
```

在BookDistribution数据库中，使用不同的服务名称和队列名称创建服务，但绑定到相同的约定，作为BookStore数据库中的服务：

```
CREATE SERVICE [//SackConsulting/BookDistributionService]
    ON QUEUE dbo.BookDistributionQueue
    ([//SackConsulting/BookOrderContract])
GO
```

现在已经创建了消息、队列、约定以及服务，可以开始在两个数据库之间使用Service Broker命令进行通信了。

20.2.7 启动对话

对话会话（dialog conversation）是在服务之间进行消息交换的操作。使用BEGIN DIALOG CONVERSATION命令来创建新的会话。所有新会话都将生成uniqueidentifier数据类型的唯一会话句柄。

语法如下：

```
BEGIN DIALOG [ CONVERSATION ] @dialog_handle
   FROM SERVICE initiator_service_name
   TO SERVICE 'target_service_name'
       [ , { 'service_broker_guid' | 'CURRENT DATABASE' } ]
   [ ON CONTRACT contract_name ]
   [ WITH
   [ { RELATED_CONVERSATION = related_conversation_handle
     | RELATED_CONVERSATION_GROUP = related_conversation_group_id } ]
   [ [ , ] LIFETIME = dialog_lifetime ]
   [ [ , ] ENCRYPTION = { ON | OFF } ] ]
```

表20-5描述了这个命令的参数。

表20-5 BEGIN DIALOG的参数

参　数	描　述
@dialog_handle	这个参数指定用来保存新对话句柄的uniqueidentifier数据类型局部变量
initiator_service_name	这个参数定义启动会话的服务
'target_service_name'	这个参数表示发起方服务进行交换消息的目标服务
'service_broker_guid' \| 'CURRENT DATABASE'	这个参数指定从sys.databases中检索到目标服务数据库的service_broker_guid。如果指定了CURRENT DATABASE，则使用当前数据库的service_broker_guid
contract_name	这个参数定义会话基于的约定名称
related_conversation_handle	这个参数表示对话所属的既有会话组的uniqueidentifier值
related_conversation_group_id	这个参数指定新对话添加到的既有会话组的uniqueidentifier值

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

参 数	描 述
dialog_lifetime	这个参数定义对话持续打开的秒数
ENCRYPTION = { ON \| OFF }	当设为ON时，对发送到发起方SQL Server实例之外的消息进行加密

589

END CONVERSATION命令结束会话的一方。停止会话的服务不再发送或接收消息。为了完成这个操作，所有服务（发起方和目标）必须停止会话。

使用SEND命令在特定的打开的会话中发送消息。在这个命令中，同时定义消息类型和消息内容。

继续在线书店的示例，随着必需的对象的确定，现在已经准备好在两个Service Broker服务之间启动会话了。

在BookStore数据库中，在批处理中执行如下的命令：

```
Use BookStore
GO

DECLARE @Conv_Handler uniqueidentifier
DECLARE @OrderMsg xml;

BEGIN DIALOG CONVERSATION @Conv_Handler
FROM SERVICE [//SackConsulting/BookOrderService]
TO SERVICE '//SackConsulting/BookDistributionService'
ON CONTRACT [//SackConsulting/BookOrderContract];

SET @OrderMsg =
'<order id="3439" customer="22" orderdate="7/15/2008">
<LineItem ItemNumber="1" ISBN="1-59059-592-0" Quantity="1" />
</order>';

SEND ON CONVERSATION @Conv_Handler
MESSAGE TYPE [//SackConsulting/SendBookOrder]
(@OrderMsg);
```

解析

在之前的批处理语句中，使用两个局部变量保存对话会话句柄和订单消息XML文档：

```
DECLARE @Conv_Handler uniqueidentifier
DECLARE @OrderMsg xml;
```

使用BEGIN DIALOG CONVERSATION命令，基于确定的约定在两个服务之间创建会话。第一个参数为@Conv_Handler局部变量：

```
BEGIN DIALOG CONVERSATION @conv_handler
```

第二行指定了用来开始会话的发起方，第三行指定了目标服务：

```
FROM SERVICE [//SackConsulting/BookOrderService]
TO SERVICE '//SackConsulting/BookDistributionService'
```

然后指定约定的名称：

```
ON CONTRACT [//SackConsulting/BookOrderContract];
```

590

局部变量@OrderMsg设置为包含订单和行项信息的XML文档：

```
SET @OrderMsg =
'<order id="3439" customer="22" orderdate="7/15/2008">
<LineItem ItemNumber="1" ISBN="1-59059-592-0" Quantity="1" />
</order>';
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

SEND ON CONVERSATION命令使用会话句柄局部变量来发送使用特定的（并且是允许的）消息类型和实际的XML消息内容的消息。命令中的第一个参数是在BEGIN DIALOG CONVERSATION命令中赋予的@Conv_Handler值：

```
SEND ON CONVERSATION @Conv_Handler
```

第二个参数是使用的消息类型，随后为局部变量中的XML消息：

```
MESSAGE TYPE [//SackConsulting/SendBookOrder]
(@OrderMsg);
```

然后将这个消息发送到BookDistribution数据库的队列中。

20.2.8 查询队列中传入的消息

继续在线书店的示例，在BookDistribution数据库中，使用SELECT查询队列来查看传入的消息。

```
USE BookDistribution
GO

SELECT message_type_name, CAST(message_body as xml) message,
queuing_order, conversation_handle, conversation_group_id
FROM dbo.BookDistributionQueue
```

查询返回如下的结果集（为了更好的阅读改变了格式）：

```
Column                  Value
Message_type_name       //SackConsulting/SendBookOrder
Message                 <order id="3439" customer="22" orderdate="7/15/2008">
                        <LineItem ItemNumber="1" ISBN="1-59059-592-0" Quantity="1" />
                        </order>
Queuing_order           0
Conversation_handle     63558054-02EE-DC11-B4A4-0003FF25C9C5
Conversation_group_id   62558054-02EE-DC11-B4A4-0003FF25C9C5
```

解析

在这个技巧中，演示了对队列使用SELECT，和操作一个表一样。返回的数据显示了消息类型、消息内容、队列序号以及指定了会话句柄和会话组的uniqueidentifier值。

20.2.9 检索并响应消息

使用RECEIVE语句从队列中读取行（消息）。不像SELECT语句，RECEIVE可以用来删除已经读取过的行。RECEIVE的结果可以填充到常规的表中，或用在局部变量中来执行其他操作，或发送其他Service Broker消息。

如果消息是xml数据类型的消息，T-SQL XQuery方法可以用来通过操作与应用程序需求相关的数据（例如，通过抽取订单ID或已订购产品的数量）来查询消息内容。

继续BookDistribution数据库中的在线书店的示例，创建新表来保存接收到的图书订单信息：

```
USE BookDistribution
GO

CREATE TABLE dbo.BookOrderReceived
   (BookOrderReceivedID int IDENTITY (1,1) NOT NULL,
    conversation_handle uniqueidentifier NOT NULL,
    conversation_group_id uniqueidentifier NOT NULL,
    message_body xml NOT NULL)
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

RECEIVE命令用来处理BookDistribution数据库中接收到的消息。这个批处理语句（一起执行的语句）执行了以下几个操作：

```
-- 声明需要保存传入消息数据的局部变量
DECLARE @Conv_Handler uniqueidentifier
DECLARE @Conv_Group uniqueidentifier
DECLARE @OrderMsg xml
DECLARE @TextResponseMsg varchar(8000)
DECLARE @ResponseMsg xml
DECLARE @OrderID int;

-- 从队列中获取消息，将接收的值赋予局部变量
RECEIVE TOP(1) @OrderMsg = message_body,
            @Conv_Handler = conversation_handle,
            @Conv_Group = conversation_group_id
FROM dbo.BookDistributionQueue;

-- 将局部变量插入表中
INSERT dbo.BookOrderReceived
(conversation_handle, conversation_group_id, message_body)
VALUES
(@Conv_Handler,@Conv_Group, @OrderMsg )

-- 对接收的值使用XQuery来进行抽取用来响应消息订单ID

SELECT @OrderID = @OrderMsg.value('(/order/@id)[1]', 'int' )

SELECT @TextResponseMsg =
      '<orderreceived id= "' +
      CAST(@OrderID as varchar(10)) +
      '"/>';

SELECT @ResponseMsg = CAST(@TextResponseMsg as xml);

-- 使用既有的会话句柄，发送响应消息回到发起方
SEND ON CONVERSATION @Conv_Handler
MESSAGE TYPE [//SackConsulting/BookOrderReceived]
(@ResponseMsg );
```

592

解析

在这个技巧中，首先创建一个表来保存传入的Service Broker消息内容。在这之后，创建了6个局部变量来保存传入的消息数据：

```
DECLARE @Conv_Handler uniqueidentifier
DECLARE @Conv_Group uniqueidentifier
DECLARE @OrderMsg xml
DECLARE @TextResponseMsg varchar(8000)
DECLARE @ResponseMsg xml
DECLARE @OrderID int;
```

然后使用RECEIVE命令从队列中返回消息。第一行中的TOP子句指定了要返回的消息的最大数量，在这个例子中为1：

```
RECEIVE TOP(1)
```

随后的几行用消息中的数据填充局部变量，与使用SELECT执行变量填充的方式相同：

```
    @OrderMsg = message_body,
@Conv_Handler = conversation_handle,
    @Conv_Group = conversation_group_id
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

RECEIVE命令的最后一行是引用消息所在的队列的FROM子句：

```
FROM dbo.BookDistributionQueue;
```

在这之后，执行了INSERT操作，将包含消息主体的值插入到了新表中：

```
INSERT dbo.BookOrderReceived
(conversation_handle, conversation_group_id, message_body)
VALUES
(@Conv_Handler,@Conv_Group, @OrderMsg )
```

使用XQuery值的方法从存储xml数据类型的数据中检索订单ID：

```
SELECT @OrderID = @OrderMsg.value
('(/order/@id)[1]', 'int' )
```

从XQuery取得值然后填充到局部变量中，以<orderreceived> XML元素嵌入值：

```
SELECT @TextResponseMsg =
      '<orderreceived id= "' +
      CAST(@OrderID as varchar(10)) +
      '"/>';
```

然后将这个变量转化为xml数据类型，为将响应发送到BookStore数据库作准备：

```
SELECT @ResponseMsg = CAST(@TextResponseMsg as xml);
```

在第一行中使用既有的会话uniqueidentifier句柄，使用SEND ON CONVERSATION来发送消息。第二行包含要发送的消息类型，以及在括号中的实际消息内容的局部变量：

```
SEND ON CONVERSATION @Conv_Handler
MESSAGE TYPE [//SackConsulting/BookOrderReceived]
(@ResponseMsg );
```

20.2.10 结束会话

会话涉及发送和接收消息。不管你的应用程序需要进行多少次迭代，通信还是可以继续。一方完成（发起方或目标）之后，就可以通过使用END CONVERSATION命令通知另一方你完成了这个会话。

在前面的技巧中，基于BookStore发送的订单消息的订单确认消息发送到了BookStore。继续在线书店的示例，创建新表来存储来自目标服务的订单确认信息：

```
USE BookStore
GO

-- 创建订单确认表
CREATE TABLE dbo.BookOrderConfirmation
    (BookOrderConfirmationID int IDENTITY (1,1) NOT NULL,
     conversation_handle uniqueidentifier NOT NULL,
     DateReceived datetime NOT NULL DEFAULT GETDATE(),
     message_body xml NOT NULL)
```

在BookStore数据库中，使用RECEIVE TOP来接收响应消息，并且将它存储在新表中。当接收到响应后，这个特殊的BookOrder的会话就完成了（对话会话应该何时结束取决于现实中的任务需求），使用END CONVERSATION命令通知目标数据库，会话的一方完成了：

```
DECLARE @Conv_Handler uniqueidentifier
DECLARE @Conv_Group uniqueidentifier
DECLARE @OrderMsg xml
DECLARE @TextResponseMsg varchar(8000);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
RECEIVE TOP(1) @Conv_Handler = conversation_handle,
          @OrderMsg = message_body
FROM dbo.BookStoreQueue

INSERT dbo.BookOrderConfirmation
(conversation_handle, message_body)
VALUES (@Conv_Handler,@OrderMsg );

END CONVERSATION @Conv_Handler;
GO
```

在BookDistribution数据库中，再次检查队列中的新消息。当会话对话结束时，将发送一条消息类型名为http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog的空消息。下一个批处理语句接收这个消息，并且如果消息类型为结束对话消息类型，则它这一方结束会话：

```
USE BookDistribution
GO

DECLARE @Conv_Handler uniqueidentifier
DECLARE @Conv_Group uniqueidentifier
DECLARE @OrderMsg xml
DECLARE @message_type_name nvarchar(256);

RECEIVE TOP(1) @Conv_Handler = conversation_handle,
          @OrderMsg = message_body,
          @message_type_name = message_type_name
FROM dbo.BookDistributionQueue

-- 双方（发起方和目标）必须都结束会话

IF
@message_type_name = 'http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog'
BEGIN
   END CONVERSATION @Conv_Handler;
END
```

594

通过查询sys.conversation_endpoints视图来检查会话的状态：

```
SELECT state_desc, conversation_handle
FROM sys.conversation_endpoints
```

这个查询返回：

```
state_desc    conversation_handle
CLOSED        237A7DD6-86FB-D911-AAF4-000FB522BF5A
```

解析

在这个技巧中，演示了如何结束一个打开的会话对话。首先，在BookStore数据库中创建一个表，来保存从BookDistribution数据库接收来的订单确认消息。在这之后，使用RECEIVE TOP (1)从BookStore-Queue中的BookDistribution获取最近的消息。然后将消息的内容插入到表BookOrderConfirmation中。然后使用END CONVERSATION和特定会话的uniqueidentifier值来结束会话：

```
END CONVERSATION @Conv_Handler;
```

结束会话会自动发送http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog的消息类型到目标数据库。回到BookDistribution数据库，再次检查队列中的新消息。使用RECEIVE TOP(1)来检索从BookStore数据库来的最近的响应。使用IF语句来验证接收的消息是否是END DIALOG请求：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
IF
@message_type_name =
'http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog'
```

如果是，则目标数据库（BookDistribution）的会话也将结束：

```
BEGIN
   END CONVERSATION @Conv_Handler;
END
```

然后通过查询sys.conversation_endpoints视图来检查会话的状态，以确认会话是否真的为CLOSED。

技巧完整地演示了一个用来发送图书订单消息到图书分发控制数据库的简单的消息交换应用程序。从发起方发送图书订单，然后发送回一个响应，并且在两个数据库上使用END CONVERSATION结束会话。当然，现实场景会涉及更多的任务，它可能又转换为其他消息类型、约定、服务和队列。理想的Service Broker任务是那些可以从异步能力中受益的、防止应用程序挂起和产生瓶颈的任务。

20.2.11 设定 Service Broker 会话的优先级

对于非常活跃的Service Broker应用程序，SQL Server 2008引入了使用CREATE BROKER PRIORITY命令对会话设定优先级的功能。使用这个功能，可以指定不重要或重要会话的优先级，以确保消息合理地发送。

为了充分利用这个功能，会话涉及的Service Broker数据库必须先启用HONOR_BROKER_PRIORITY选项。在这个技巧的第一个查询中，将演示Service Broker应用程序的两个数据库中启用这个选项：

```
ALTER DATABASE BookStore
SET HONOR_BROKER_PRIORITY ON

ALTER DATABASE BOOKDistribution
SET HONOR_BROKER_PRIORITY ON
```

可以通过查询sys.databases来确认数据库的更改：

```
SELECT name, is_honor_broker_priority_on
FROM sys.databases
WHERE name IN ('BookStore', 'BookDistribution')
```

这个查询返回：

```
name            is_honor_broker_priority_on
BookStore       1
BookDistribution    1
```

现在可以使用命令基于本地服务名称远程服务名称（关于远程服务更多信息稍后讲解）和约定名称来定义指定会话的优先级。优先级值的范围从1到10，1是最低优先级，10是最高优先级。会话优先级默认值为5。

在这个技巧中，将演示基于指定约定和本地服务将会话优先级设置为10：

```
USE BookStore
GO

CREATE BROKER PRIORITY Conv_Priority_BookOrderContract_BookOrderService
FOR CONVERSATION
SET (CONTRACT_NAME = [//SackConsulting/BookOrderContract],
   LOCAL_SERVICE_NAME = [//SackConsulting/BookOrderService],
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
   REMOTE_SERVICE_NAME = ANY,
   PRIORITY_LEVEL = 10)
```

注意，在这里REMOTE_SERVICE_NAME使用了ANY值。ANY可以用作约定、本地服务或远程服务的值，在这个例子中表示将优先级关联到与Service Broker端点相关联的所有服务或约定上。

下一步，可以通过查询sys.conversation_priorities目录视图来确认刚刚创建的代理优先级：

```
SELECT name, priority, service_contract_id,
       local_service_id,remote_service_name
FROM sys.conversation_priorities cp
```

596

这个查询返回：

```
name                                priority   service_contract_id   local_service_id   remote_service_name
Conv_Priority_BookOrderContract_    10         65536                 65536              NULL
BookOrderService
```

如果希望包含服务和约定名称，可以将服务和从sys.conversation_priorities返回的约定ID与sys.service_contract和sys.services目录视图关联起来。

接着，还将在BookDistribution数据库中创建代理优先级定义，来覆盖双向通讯的优先级：

```
USE BookDistribution
GO

CREATE BROKER PRIORITY Conv_Priority_BookOrderContract_BookDistributionService
FOR CONVERSATION
SET (CONTRACT_NAME = [//SackConsulting/BookOrderContract],
   LOCAL_SERVICE_NAME = [//SackConsulting/BookDistributionService],
   REMOTE_SERVICE_NAME = ANY,
   PRIORITY_LEVEL = 10)
```

可以使用ALTER BROKER PRIORITY命令修改既有的代理优先级。在下一查询中，将远程服务的设置从ANY更改为指定远程服务：

```
USE BookStore
GO

ALTER BROKER PRIORITY Conv_Priority_BookOrderContract_BookOrderService
FOR CONVERSATION
SET (REMOTE_SERVICE_NAME = '//SackConsulting/BookDistributionService')
```

下面使用ALTER BROKER PRIORITY命令修改代理优先级：

```
ALTER BROKER PRIORITY Conv_Priority_BookOrderContract_BookOrderService
FOR CONVERSATION
SET (PRIORITY_LEVEL = 9)
```

使用DROP BROKER PRIORITY命令移除代理优先级定义，如下所示：

```
DROP BROKER PRIORITY Conv_Priority_BookOrderContract_BookOrderService
```

解析

在这个技巧中，演示了如何定义Service Broker会话优先级。在操作它之前，要先启用双方参与数据库中的HONOR_BROKER_PRIORITY数据库选项。启用这个选项允许Service Broker对话的消息基于所有代理优先级规则进行发送。

在启用了数据库选项之后，使用CREATE BROKER PRIORITY命令定义基于指定会话属性的更高优先级。一点点看执行的代码，第一行定义了代理优先级对象的名称：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE BROKER PRIORITY Conv_Priority_BookOrderContract_BookOrderService
FOR CONVERSATION
```

下一步，SET语句在括号内指定约定名称、本地服务、远程服务和优先级值：

597

```
SET (CONTRACT_NAME = [//SackConsulting/BookOrderContract],
   LOCAL_SERVICE_NAME = [//SackConsulting/BookOrderService],
   REMOTE_SERVICE_NAME = ANY,
   PRIORITY_LEVEL = 10)
```

在创建完毕之后，所有在BookOrderContract约定和BookOrderService本地服务的会话对话之中的消息发送优先级都被设置为10（高于默认值5）。

然后，为目标服务创建了第二个代理优先级来处理与发起方服务会话对话交互。最后，演示了如何使用ALTER BROKER PRIORITY修改既有的代理优先级定义，还演示了如何使用DROP BROKER PRIORITY来移除它。

对于非常活跃的Service Broker应用程序，使用会话优先级可以控制约定和服务中的多种会话的重要程度。

20.3 创建用来处理消息的存储过程

在前面的几个技巧中，使用了临时的T-SQL批处理来处理从队列传入的消息。当然，也可以使用存储过程或外部应用程序创建服务程序来自动地激活并处理队列中的消息。使用CREATE QUEUE和ALTER QUEUE选项，也可以指定可以激活并处理在同一队列中传入的消息的、同时执行的相同服务程序的数量。

创建书店存储过程

使用前面技巧中既有的对象来设置存储过程应用程序，这个示例创建了用来处理在dbo.BookDistributionQueue上传入的消息的存储过程。这个过程使用了前面技巧使用的几个RECEIVE和SEND命令，并修改为存储过程形式：

```
USE BookDistribution
GO

CREATE PROCEDURE dbo.usp_SB_ReceiveOrders
AS

DECLARE @Conv_Handler uniqueidentifier
DECLARE @Conv_Group uniqueidentifier
DECLARE @OrderMsg xml
DECLARE @TextResponseMsg varchar(8000)
DECLARE @ResponseMsg xml
DECLARE @Message_Type_Name nvarchar(256);
DECLARE @OrderID int;

-- 当发生运行时错误时，XACT_ABORT自动回滚事务
SET XACT_ABORT ON

BEGIN TRAN;

   RECEIVE TOP(1) @OrderMsg = message_body,
               @Conv_Handler = conversation_handle,
               @Conv_Group = conversation_group_id,
               @Message_Type_Name = message_type_name
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

598
```
    FROM dbo.BookDistributionQueue;

    IF @Message_Type_Name = '//SackConsulting/SendBookOrder'
    BEGIN
       INSERT dbo.BookOrderReceived
       (conversation_handle, conversation_group_id, message_body)
       VALUES
       (@Conv_Handler,@Conv_Group, @OrderMsg )

    SELECT @OrderID = @OrderMsg.value('(/order/@id)[1]', 'int' )

    SELECT @TextResponseMsg =

          '<orderreceived id= "' +
          CAST(@OrderID as varchar(10)) +
          '"/>';

    SELECT @ResponseMsg = CAST(@TextResponseMsg as xml);

    SEND ON CONVERSATION @Conv_Handler
    MESSAGE TYPE [//SackConsulting/BookOrderReceived]
    (@ResponseMsg );
END

IF @Message_Type_Name = 'http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog'
BEGIN
   END CONVERSATION @Conv_Handler;
END

COMMIT TRAN

GO
```

这个过程包含处理// SackConsulting /SendBookOrder 和 http:// schemas.microsoft.com /SQL/ServiceBroker/EndDialog消息类型的逻辑。如果发送了后者，特定会话句柄的特定会话将结束。如果接收到图书订单消息类型，它的信息将插入到表中，并且返回订单确认信息。

可以使用ALTER QUEUE命令修改既有的队列。这个命令使用与CREATE QUEUE相同的选项，它允许你改变队列的状态与保持期、待激活的存储过程、队列读取存储过程实例的最大数量以及过程的安全模式约定。

ALTER QUEUE包括一个额外的参数DROP，它用来删除队列的所有存储过程激活设置。

使用ALTER QUEUE命令将存储过程绑定到既有的队列：

```
ALTER QUEUE dbo.BookDistributionQueue
WITH ACTIVATION (STATUS = ON,
            PROCEDURE_NAME = dbo.usp_SB_ReceiveOrders,
            MAX_QUEUE_READERS = 2,
            EXECUTE AS SELF)
```

指定过程名称，随后是可以独立处理队列中不同消息的同一存储过程同时执行的最大数量。

599 为了测试BookStore数据库的新服务程序，开始了一个会话并设置了新的顺序：

```
Use BookStore
GO

DECLARE @Conv_Handler uniqueidentifier
DECLARE @OrderMsg xml;

BEGIN DIALOG CONVERSATION @conv_handler
FROM SERVICE [//SackConsulting/BookOrderService]
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
TO SERVICE '//SackConsulting/BookDistributionService'
ON CONTRACT [//SackConsulting/BookOrderContract];

SET @OrderMsg =
'<order id="3490" customer="29" orderdate="7/22/2008">
<LineItem ItemNumber="1" ISBN="1-59059-592-0" Quantity="2" />
</order>';

SEND ON CONVERSATION @Conv_Handler
MESSAGE TYPE [//SackConsulting/SendBookOrder]
(@OrderMsg);
```

如果在目标队列上有存储过程被执行，并且激活新的接收到的消息，那么应该已经有订单确认消息返回到dbo.BookStoreQueue：

```
SELECT conversation_handle, CAST(message_body as xml) message
FROM dbo.BookStoreQueue
```

这个查询返回如下的结果：

```
conversation_handle                  message
20E768EB-8EFB-D911-AAF4-000FB522BF5A  <orderreceived id="3490" />
```

解析

在这个技巧中，创建了处理队列中消息的存储过程。使用ALTER QUEUE命令将这个存储过程绑定到队列上。这个命令的第一个参数是要修改的队列名称：

```
ALTER QUEUE dbo.BookDistributionQueue
```

WITH ACTIVATION子句首先指定了新应用程序（存储过程）的状态可以接收新消息：

```
WITH ACTIVATION (STATUS = ON,
```

下一步，指定绑定到队列的存储过程的名称：

```
PROCEDURE_NAME = dbo.usp_SB_ReceiveOrders,
```

使用MAX QUEUE READERS选项指定存储过程应用程序同时执行的最大数量为2：

```
MAX_QUEUE_READERS = 2,
```

指定EXECUTE AS参数为SELF，意思是存储过程将以与执行ALTER QUEUE命令的主体的相同的权限来执行：

```
EXECUTE AS SELF)
```

当队列STATUS = ON并且队列中到达新消息时，执行存储过程来处理传入的消息。你可以使用内部存储过程或外部应用程序来处理队列中传入的消息。但是使用存储过程的益处是，它们提供了处理消息、自动地执行所有需要的响应和相关业务任务的简单的封装好的组件。

20.4 远程服务器 Service Broker 的实现

为了演示Service Broker程序基本的设置，本章中的示例使用了同一SQL Server实例中的两个数据库。但在大多数情况下，会将Service Broker设置为使用存在于两个或更多的SQL Server实例上的数据库进行工作。核心的组件还是一样，只不过要实现跨服务器通信还需要一些其他步骤。可以通过使用Windows身份验证或基于证书的身份验证（马上就会演示）来完成跨服务器通信。这些步骤将在以下一组技巧中演示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下面是启用跨SQL Server实例的Service Broker通信的大体步骤。

(1) 启用传输安全模式。Service Broker中的传输安全模式应用于两个SQL Server实例之间的网络连接，在它们之间启用或限制加密的通信。这需要在两个SQL Server实例中的系统数据库master中设置，而且正如你将会看到的，其中包括创建端点、证书、登录名和用户。

(2) 启用对话安全模式。Service Broker的对话安全模式提供了身份验证、授权以及对话会话的加密。在用于Service Broker执行的实际数据库中，创建证书以及它们在SQL Server实例之间交换的公钥。创建的用户不会关联到登录名，但是会提供从另一SQL Server实例的公钥上创建的证书的身份验证。

(3) 创建路由。Service Broker使用路由来确定服务的位置，确定它是本地的还是远程的。

(4) 创建远程服务绑定。远程服务绑定用来映射打开远程Service Broker服务会话的安全凭证。

在这个跨服务器的场景中，在线书店Service Broker程序将使用JOEPROD SQL Server实例中的BookStore数据库以及JOEPROD\NODE2 SQL Server实例中的BookDistribution数据库。使用前面一组技巧中的对象来演示这个功能。首先，我们执行删除操作（如果你确实一路做下来），在JOEPROD上删除示例数据库并且重新创建BookStore数据库，并在JOEPROD\NODE2上删除示例数据库并且重新创建BookDistribution数据库。然后在JOEPROD实例的BookStore数据库上创建下面的对象并配置它的设置：

```
USE master
GO

-- 启用数据库的Service Broker

ALTER DATABASE BookStore SET ENABLE_BROKER
GO

ALTER DATABASE BookStore SET TRUSTWORTHY ON
GO

USE BookStore
GO

-- 创建消息

CREATE MESSAGE TYPE [//SackConsulting/SendBookOrder]
VALIDATION = WELL_FORMED_XML
GO

CREATE MESSAGE TYPE [//SackConsulting/BookOrderReceived]
VALIDATION = WELL_FORMED_XML
GO

-- 创建约定

CREATE CONTRACT
    [//SackConsulting/BookOrderContract]
    ( [//SackConsulting/SendBookOrder]
          SENT BY INITIATOR,
      [//SackConsulting/BookOrderReceived]
          SENT BY TARGET
    )
GO

-- 创建队列

CREATE QUEUE BookStoreQueue
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
WITH STATUS=ON
GO

-- 创建服务

CREATE SERVICE [//SackConsulting/BookOrderService]
    ON QUEUE dbo.BookStoreQueue
    ([//SackConsulting/BookOrderContract])
GO
```

在JOEPROD\NODE2实例的BookDistribution数据库上创建下面的对象：

```
USE master
GO

IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'BookDistribution')

CREATE DATABASE BookDistribution
GO

-- 启用数据库的Service Broker功能

ALTER DATABASE BookDistribution SET ENABLE_BROKER
GO

ALTER DATABASE BookDistribution SET TRUSTWORTHY ON
GO

USE BookDistribution
GO

-- 创建消息

CREATE MESSAGE TYPE [//SackConsulting/SendBookOrder]
VALIDATION = WELL_FORMED_XML
GO

CREATE MESSAGE TYPE [//SackConsulting/BookOrderReceived]
VALIDATION = WELL_FORMED_XML
GO

-- 创建约定

CREATE CONTRACT
    [//SackConsulting/BookOrderContract]
    ( [//SackConsulting/SendBookOrder]
          SENT BY INITIATOR,
      [//SackConsulting/BookOrderReceived]
          SENT BY TARGET
  )
GO

-- 创建队列

CREATE QUEUE BookDistributionQueue
WITH STATUS=ON
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
-- 创建服务
CREATE SERVICE [//SackConsulting/BookDistributionService]
    ON QUEUE dbo.BookDistributionQueue
    ([//SackConsulting/BookOrderContract])
GO
```

20.4.1 启用传输安全模式

Service Broker中的传输安全模式应用于两个SQL Server实例间的网络连接，启用或限制它们之间的加密通信。传输安全模式位于SQL Server实例级别，因此这个技巧演示了在两个SQL Server实例的数据库master中创建对象。可以从两种形式的传输安全模式中进行选择：Windows身份验证或基于证书的安全模式。

这个技巧包括前面章节中学习的涉及操作对象的几个步骤。所有这些步骤需要两个SQL Server实例（在这个示例中包括JOEPROD和JOEPROD\NODE2）都是活动的。对于这个技巧，因为传输安全模式应用在SQL Server实例本身，所以在这个技巧中只可以使用系统数据库master，不可以使用实际的用户数据库。

首先在所有SQL Server实例的系统数据库master中创建数据库主密钥。创建的目的是加密用在基于证书的传输安全模式的证书：

603

```
-- 在JOEPROD上执行
USE master
GO

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '1294934A!'

-- 在JOEPROD\NODE2上执行
USE master
GO

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '1294934B!'
```

下一步，在所有SQL Server实例的系统数据库master中创建新证书：

```
-- 在JOEPROD上执行
CREATE CERTIFICATE JOEPRODMasterCert
   WITH SUBJECT = 'JOEPROD Transport Security SB',
   EXPIRY_DATE = '10/1/2010'
GO

-- 在JOEPROD\NODE2上执行
CREATE CERTIFICATE Node2MasterCert
   WITH SUBJECT = 'Node 2 Transport Security SB',
   EXPIRY_DATE = '10/1/2010'
GO
```

下一步，将所有这些证书备份到文件。然后复制公钥备份文件，用来在另一SQL Server实例的数据库master中创建证书（稍后会在这个技巧中提到）：

```
-- 在JOEPROD上执行
BACKUP CERTIFICATE JOEPRODMasterCert
TO FILE = 'C:\Apress\JOEPRODMasterCert.cer'
GO

-- 在JOEPROD\NODE2上执行
BACKUP CERTIFICATE Node2MasterCert
TO FILE = 'C:\Apress\Node2MasterCert.cer'
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在每个SQL Server实例中创建Service Broker端点，它们都使用基于身份验证的证书并且需要加密通信。

```
-- 在JOEPROD上执行

CREATE ENDPOINT SB_JOEPROD_Endpoint
STATE = STARTED
AS TCP
(LISTENER_PORT = 4020)
FOR SERVICE_BROKER
(AUTHENTICATION = CERTIFICATE JOEPRODMasterCert,
  ENCRYPTION = REQUIRED)
GO

-- 在JOEPROD\NODE2上执行
CREATE ENDPOINT SB_NODE2_Endpoint
STATE = STARTED
AS TCP
(LISTENER_PORT = 4021)
FOR SERVICE_BROKER
(AUTHENTICATION = CERTIFICATE Node2MasterCert,
  ENCRYPTION = REQUIRED)
GO
```


在两个SQL Server实例中，在系统数据库master中创建新的、用作到另一SQL Server实例的远程连接的登录名和用户：

```
-- 在JOEPROD上执行
CREATE LOGIN SBLogin
    WITH PASSWORD = 'Used4TransSec'
GO

CREATE USER SBUser
    FOR LOGIN SBLogin
GO

-- 在JOEPROD\NODE2上执行

CREATE LOGIN SBLogin
    WITH PASSWORD = 'Used4TransSec'
GO

CREATE USER SBUser
    FOR LOGIN SBLogin
GO
```

下一步，授予到关联的端点的CONNECT权限给两个SQL Server实例的登录名：

```
-- 在JOEPROD上执行

GRANT CONNECT ON Endpoint::SB_JOEPROD_Endpoint TO SBLogin
GO

-- 在JOEPROD\NODE2上执行

GRANT CONNECT ON Endpoint::SB_NODE2_Endpoint TO SBLogin
GO
```

在两个SQL Server实例中，基于在另一SQL Server实例中创建的证书备份创建新的证书。对前面

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

步骤中新创建的登录名和用户授予这个证书的身份验证权限：

```
-- 在JOEPROD上执行

CREATE CERTIFICATE Node2MasterCert
AUTHORIZATION SBUser
FROM FILE = 'C:\Apress\Node2MasterCert.cer'
GO

-- 在JOEPROD\NODE2上执行

CREATE CERTIFICATE JOEPRODMasterCert
AUTHORIZATION SBUser
FROM FILE = 'C:\Apress\JOEPRODMasterCert.cer'
GO
```

605

解析

在这个技巧中，介绍了通过证书建立传输安全模式所需要的各种步骤。首先，创建用来加密证书的数据库主密钥（正如Server Broker端点的需求——假设使用基于证书的安全模式，则证书不能是加密后的密码）。

在两个SQL Server实例中创建证书，然后备份并复制到另一SQL Server实例。交换公钥将在这部分稍后用到。同时，在所有SQL Server实例中创建Service Broker端点，并对其进行配置以允许从另一基于证书安全模式的服务器进行访问。

之后，在两个SQL Server实例中创建登录名和用户。授予登录名到端点的CONNECT权限。但这还不足以启用连通性，因为用户也必须访问用在另一SQL Server实例的证书的公钥。为了与另一服务器交换密钥，所以授予了这个权限。然后将新证书绑定到两个实例上新创建的用户上。因为用户拥有对另一SQL Server实例的证书的权限，并且因为端点是基于证书的，所以SQL Server实例将让加密传输安全模式互相访问。

这仅仅是允许跨服务器Service Broker通信的一半条件。下一步是用户数据库级别的对话安全模式，我将在下一技巧中演示它。

20.4.2 启用对话安全模式

尽管传输安全模式在SQL Server实例级别控制通信，Service Broker的对话安全模式提供了身份验证、授权以及对对话会话的加密。像前面的技巧一样，设置对话安全模式涉及一些小步骤，其中涉及的很多命令已经在前面一些章节中讲到了。

这些技巧将在JOEPROD SQL Server实例上的BookStore数据库和NODE2 SQL Server实例上的BookDistribution数据库上进行实践。在所有SQL Server实例上创建证书（它需要在这部分开始时创建的数据库主密钥）。然后，要在SQL Server实例间交换证书，这与前面技巧中传输安全模式的操作类似：

```
-- 在JOEPROD上执行
USE BookStore
GO

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '1294934A!'
GO

CREATE CERTIFICATE BookStoreCert
   WITH SUBJECT = 'BookStore SB cert',
   EXPIRY_DATE = '10/1/2010'
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
-- 在NODE2上执行
USE BookDistribution
GO

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '1294934B!'
GO

CREATE CERTIFICATE BookDistributionCert
   WITH SUBJECT = 'BookDistributionCert SB cert',
   EXPIRY_DATE = '10/1/2010'
GO
```

606

下一步，将所有数据库的证书备份到文件：

```
-- 在JOEPROD上执行
USE BookStore
GO

BACKUP CERTIFICATE BookStoreCert
TO FILE = 'C:\Apress\BookStoreCert.cer'
GO

-- 在NODE2上执行
USE BookDistribution
GO

BACKUP CERTIFICATE BookDistributionCert
TO FILE = 'C:\Apress\BookDistributionCert.cer'
GO
```

20

在这之后，在所有数据库上创建用户。任何一个用户都不会关联到登录名。然后，这个用户将映射到另一SQL Server实例的公有证书上：

```
-- 在JOEPROD上执行
USE BookStore
GO

CREATE USER BookDistributionUser
WITHOUT LOGIN
GO

-- 在NODE2上执行
USE BookDistribution
GO

CREATE USER BookStoreUser
WITHOUT LOGIN
GO
```

下一步，在所有基于另一数据库的证书公钥的数据库上创建新证书。对每个数据库中新创建的用户授予到这个证书的权限：

```
-- 在JOEPROD上执行
USE BookStore
GO

CREATE CERTIFICATE BookDistributionCert
AUTHORIZATION BookDistributionUser
FROM FILE = 'C:\Apress\BookDistributionCert.cer'
GO
```

607

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
-- 在NODE2上执行
USE BookDistribution
GO

CREATE CERTIFICATE BookStoreCert
AUTHORIZATION BookStoreUser
FROM FILE = 'C:\Apress\BookStoreCert.cer'
GO
```

最后，两个数据库的用户都需要到关联的Service Broker服务的SEND权限：

```
-- 在JOEPROD上执行
USE BookStore
GO

GRANT SEND ON
SERVICE::[//SackConsulting/BookOrderService] TO BookDistributionUser
GO

-- 在NODE2上执行
USE BookDistribution
GO

GRANT SEND ON
SERVICE::[//SackConsulting/BookDistributionService]
TO BookStoreUser
```

解析

在这个技巧中，演示了设置对话安全模式，它控制身份验证、授权以及在Service Broker应用程序中两个用户定义数据库之间的加密。

第一步包括在每个数据库中创建数据库主密钥，它将用来隐式加密在BookStore数据库和BookDistribution中创建的证书。创建证书之后，为所有数据库主密钥进行备份，并将关联的文件复制到另一服务器。

在这之后，在没有关联的登录名的所有数据库中创建新用户。然后在所有基于另一数据库证书的数据库中创建新证书。创建证书包括指定每个数据库中新用户的AUTHORIZATION子句。

最后，对两个用户授予将消息SEND到关联的Service Broker服务的权限。

至此，Service Broker应用程序只需要两步就能跨SQL Server实例通信了。

20.4.3　创建路由和远程服务绑定

一旦设置了传输和对话安全对象，这个分布式的在线书店的示例的下一个步骤是设置路由和远程服务绑定。

Service Broker使用路由确定服务器的位置是本地的还是远程的。使用CREATE ROUTE命令创建路由。语法如下：

```
CREATE ROUTE route_name
[ AUTHORIZATION owner_name ]
WITH
   [ SERVICE_NAME = 'service_name', ]
[ BROKER_INSTANCE = 'broker_instance_identifier' , ]
[ LIFETIME = route_lifetime , ]
ADDRESS = 'next_hop_address'
[ , MIRROR_ADDRESS = 'next_hop_mirror_address' ]
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表20-6描述了这个命令的参数。

表20-6 CREATE ROUTE的参数

参 数	描 述
route_name	这个选项定义新路由的名称
AUTHORIZATION owner_name	这个选项指定路由拥有者的数据库主体
SERVICE_NAME = 'service_name'	这个选项定义路由指向的远程服务的名称
BROKER_INSTANCE='broker_instance_identifier'	这个选项指定承载目标服务的数据库的service_broker_guid（sys.data-bases中）
LIFETIME = route_lifetime	这个选项指定SQL Server在路由表中保留路由的过期时间（秒）
ADDRESS = 'next_hop_address'	这个选项定义服务驻留的SQL Server实例的DNS、NetBios或TCP/IP地址。也包括使用TCP://{ dns_name \| netbios_name \| ip_address } : port_number语法的Service Broker端点的端口号
MIRROR_ADDRESS='next_hop_mirror_address'	如果使用数据库镜像，这个选项可以使用TCP://{ dns_name \| netbios_name \| ip_address } : port_number语法指定镜像数据库的地址

在这个技巧的示例中，在JOEPROD上创建指向NODE2 Service Broker端点的路由（监听4021端口），并且引用BookDistribution数据库的//SackConsulting/BookDistributionService服务：

```
-- 在JOEPROD上执行
USE BookStore
GO

CREATE ROUTE Route_BookDistribution
WITH SERVICE_NAME = '//SackConsulting/BookDistributionService',
ADDRESS = 'TCP://192.168.0.105:4021'
GO
```

在NODE2上，创建指向JOEPROD Service Broker端点的路由（监听4020端口），并且引用BookStore数据库的//SackConsulting/BookStoreService服务：

```
-- 在NODE2上执行
USE BookDistribution
GO

CREATE ROUTE Route_BookStore
WITH SERVICE_NAME = '//SackConsulting/BookOrderService',
ADDRESS = 'TCP://192.168.0.105:4020'
GO
```

使用远程服务器绑定来映射用来打开远程Service Broker服务会话的安全凭证。具体来说，是使用远程服务绑定到前面技巧中创建的用户上（映射到证书上的那一个）。使用CREATE REMOTE SERVICE BINDING命令创建远程服务绑定。语法如下：

```
CREATE REMOTE SERVICE BINDING binding_name
   [ AUTHORIZATION owner_name ]
   TO SERVICE 'service_name'
   WITH USER = user_name [ , ANONYMOUS = { ON | OFF } ]
```

表20-7描述了这个命令的参数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表20-7 CREATE REMOTE SERVICE BINDING的参数

参 数	描 述
binding_name	这个选项指定新远程服务绑定的名称
AUTHORIZATION owner_name	这个选项定义绑定的数据库主体的拥有者
service_name	这个选项表示要绑定的远程服务的名称
USER = user_name	这个选项指定映射到远程服务的证书的数据库用户
ANONYMOUS = { ON \| OFF }	当这个选项为ON时，匿名身份验证在公开的固定数据库角色的上下文中用来连接到远程数据库

在JOEPROD上的这个示例中，使用映射到BookDistribution数据库的公开证书的BookStore用户，生成了从BookStore到//SackConsulting/BookDistributionService服务的绑定：

```
USE BookStore
GO

CREATE REMOTE SERVICE BINDING BookDistributionBinding
    TO SERVICE '//SackConsulting/BookDistributionService'
    WITH USER = BookDistributionUser
GO
```

在NODE2，在BookDistribution数据库中生成相似的绑定，只是这次指向//SackConsulting/BookOrderService服务：

```
USE BookDistribution
GO

CREATE REMOTE SERVICE BINDING BookStoreBinding
    TO SERVICE '//SackConsulting/BookOrderService'
    WITH USER = BookStoreUser
GO
```

设置好路由和绑定之后，现在可以测试从JOEPROD服务器的BookStore数据库发送远程消息到NODE2
610 服务器的BookDistribution数据库了：

```
Use BookStore
GO

DECLARE @Conv_Handler uniqueidentifier
DECLARE @OrderMsg xml;

BEGIN DIALOG CONVERSATION @Conv_Handler
FROM SERVICE [//SackConsulting/BookOrderService]
TO SERVICE '//SackConsulting/BookDistributionService'
ON CONTRACT [//SackConsulting/BookOrderContract];

SET @OrderMsg =
'<order id="3439" customer="22" orderdate="9/25/2008">
<LineItem ItemNumber="22" ISBN="1-59059-592-0" Quantity="10" />
</order>';

SEND ON CONVERSATION @Conv_Handler
MESSAGE TYPE [//SackConsulting/SendBookOrder]
(@OrderMsg);
```

移动到服务器NODE2和BookDistribution数据库，检查队列中传入的消息：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
USE BookDistribution
GO

SELECT message_type_name, CAST(message_body as xml) message,
queuing_order, conversation_handle, conversation_group_id
FROM dbo.BookDistributionQueue
```

这个查询返回如下的结果集（为了便于理解做了删减）：

```
Column                  Value
Message_type_name       //SackConsulting/SendBookOrder
Message                 <order id="3439" customer="22" orderdate="9/25/2008">
                        <LineItem ItemNumber="22" ISBN="1-59059-592-0" Quantity="10" />
                        </order>
Queuing_order           0
Conversation_handle     8150EB31-07EE-DC11-B4A4-0003FF25C9C5
Conversation_group_id   8050EB31-07EE-DC11-B4A4-0003FF25C9C5
```

解析

在这个技巧中，首先在两个SQL Server实例上创建路由。所有路由都包含了另一SQL Server实例的服务名称、连接到它的地址以及Service Broker端点的端口号。

在这之后，在两个SQL Server实例上创建用来将本地数据库用户（关联到另一SQL Server实例的公钥证书的数据库用户）映射到远程服务的远程服务绑定。

完成了这项内容后，从BookStore数据库发送消息到远程NODE2服务器的BookDistribution数据库。 611

20.5 事件通知

事件通知是集成到Service Broker的功能，使用它可以在SQL Server实例中异步捕捉SQL事件，将事件信息路由到特定的队列中。只需最小的系统开销，就可以跟踪发生在SQL Server实例的事件，比如用户登录、存储过程重编译、权限修改、对象处理（例如对数据库、程序集、角色或表的CREATE/ALTER/DROP事件）。

不像创建你自己的Service Broker应用程序，因为可以控制发起方组件，使用事件通知后只需要创建队列和Service Broker组件。用来捕捉和发送事件通知的发起方组件（消息类型和约定）已经内建在SQL Server中。

下一技巧将演示相关的操作。

捕获登录命令

在这个技巧中，演示了如何通过使用事件通知捕捉所有在SQL Server实例中执行的CREATE LOGIN、ALTER LOGIN或DROP LOGIN命令。创建事件通知的命令如下：

```
CREATE EVENT NOTIFICATION event_notification_name
ON { SERVER | DATABASE | QUEUE queue_name }
[ WITH FAN_IN ]
FOR { event_type | event_group } [ ,...n ]
TO SERVICE 'broker_service' , { 'broker_instance_specifier' |
 'current database'}
```

表20-8描述了这个命令的参数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表20-8 CREATE EVENT NOTIFICATION的参数

参 数	描 述
event_notification_name	这个参数定义了新事件通知的名称
SERVER \| DATABASE \| QUEUE queue_name	这3个参数定义了事件通知的范围，当在特定的SQL Server实例（SERVER）、当前数据库（DATABASE）或特定队列（QUEUE queue_name）上发生事件时将引发通知
WITH FAN_IN	这个参数配置SQL Server，对于创建在同一事件、同一主体并且同一服务和broker_instance_specifier的事件通知的每个事件只发送一个消息
{ event_type \| event_group } [,...n]	event_type是要监视的T-SQL DDL、Service Broker或SQL跟踪事件类型。event_group是预定义事件类型组——并且当指定它时，组的任何成员将引发一个事件通知。比如事件组DDL_LOGIN_EVENTS，它包含CREATE LOGIN、ALTER LOGIN和DROP LOGIN事件
'broker_service',{'broker_instance_specifier' \| 'current database'}	broker_service参数是接收事件通知数据的代理服务的名字。broker_instance_specifier是目标数据库的service_broker_guid（在sys.databases中），'current database'用来指定当前数据库guid

612

示例从名为EventTracking的数据库中开始，在其中创建新队列来保存事件信息：

```
IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'EventTracking')

CREATE DATABASE EventTracking
GO

USE EventTracking
GO

CREATE QUEUE SQLEventQueue
WITH STATUS=ON;
GO
```

下一步，在队列上创建新服务，关联到内建的事件通知约定：

```
CREATE SERVICE [//JOEPROD/TrackLoginModificationService]
ON QUEUE SQLEventQueue
([http://schemas.microsoft.com/SQL/Notifications/PostEventNotification]);
GO
```

下一步，为了检索数据库EventTracking的service_broker_guid（将用在CREATE EVENT NOTIFICATION命令中），对系统目录视图sys.databases执行查询：

```
select service_broker_guid
from sys.databases
WHERE name = 'EventTracking'
```

这个查询返回：

```
service_broker_guid
C72069CD-ACBA-4EA8-80BB-5CC6FF3A40AA
```

下一步，使用SERVER范围创建事件通知来跟踪SQL Server实例中所有登录名的创建、修改或删除操作：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE EVENT NOTIFICATION EN_LoginEvents
ON SERVER
FOR CREATE_LOGIN, ALTER_LOGIN, DROP_LOGIN
TO SERVICE '//JOEPROD/TrackLoginModificationService',
'C72069CD-ACBA-4EA8-80BB-5CC6FF3A40AA';
```

613

测试新的事件通知，创建一个新登录名：

```
CREATE LOGIN TrishelleN WITH PASSWORD = 'AR!3i2ou4'
GO
```

下一步，可以使用SELECT或RECEIVE（RECEIVE不像SELECT，它会删除队列中的事件消息）查询队列：

```
SELECT CAST(message_body as xml) EventInfo
FROM dbo.SQLEventQueue
```

这个查询将返回关于登录名事件的XML信息，包括添加的登录名名称以及执行添加操作的登录名：

```
<EVENT_INSTANCE>
  <EventType>CREATE_LOGIN</EventType>
  <PostTime>2008-03-09T12:39:09.493</PostTime>
  <SPID>53</SPID>
  <ServerName>CAESAR\AUGUSTUS</ServerName>
  <LoginName>CAESAR\Administrator</LoginName>
  <ObjectName>TrishelleN</ObjectName>
  <ObjectType>LOGIN</ObjectType>
  <DefaultLanguage>us_english</DefaultLanguage>
  <DefaultDatabase>master</DefaultDatabase>
  <LoginType>SQL Login</LoginType>
  <SID>McTRGulDYE2R8FTJYClNlw==</SID>
  <TSQLCommand>
    <SetOptions ANSI_NULLS="ON" ANSI_NULL_DEFAULT="ON" ANSI_PADDING="ON"
                QUOTED_IDENTIFIER="ON" ENCRYPTED="FALSE" />
    <CommandText>CREATE LOGIN TrishelleN WITH PASSWORD = '******'
</CommandText>
 </TSQLCommand>
</EVENT_INSTANCE>
```

20

解析

在这个技巧中，演示了通过执行如下步骤来设置事件通知。

(1) 在既有数据库中创建新队列。

(2) 创建绑定到新队列和内建事件通知约定的新服务。

(3) 使用CREATE EVENT NOTIFICATION命令跟踪一个或多个事件或事件组。

事件通知功能提供了跟踪SQL Server实例、数据库或Server Broker应用程序级别中的活动的低开销的方法。正如你在示例中看到的，跟踪事件只需要非常少的代码。这个新功能对于IT安全或业务级别审计需求来说特别有用。例如，当捕获到登录名创建事件时，也会被捕获到创建它的用户，以及登录名的类型（SQL登录名）、默认数据库、语言和新登录名的安全标识符。

614

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第21章 配置和查看SQL Server的选项

尽管SQL Server在幕后自动地维护和调节大多数设置和配置，但是仍然有一些管理员可以配置的数据库选项。在这个简短的章节中，我们将展示使用T-SQL查看和配置SQL Server设置的技巧。

注解 要了解SERVERPROPERTY、@@SERVERNAME以及其他SQL Server实例级的函数，请看第8章。

查看 SQL Server 的配置情况

SQL Server配置控制一系列的行为，从管理内存的方法到索引默认的填充因子。尽管有效的配置值是可变的，但基于要修改的选项，可以使用系统存储过程sp_configure来查看或做出修改：

sp_configure的语法如下：

```
sp_configure [ [ @configname = ] 'option_name'
    [ , [ @configvalue = ] 'value' ] ]
```

表21-1简要地描述了参数。

表21-1 sp_configure的参数

参 数	描 述
[@configname =] 'option_name'	要配置的SQL Server选项的名称
[@configvalue =]'value'	要设置的SQL Server选项的值

使用sp_configure修改和查询SQL Server实例的配置情况。也可以使用系统目录视图sys.configurations来查询配置。可以像查询任何普通表一样查询视图sys.configurations，它返回所有配置名称、SQL Server实例使用的值、配置设置的描述、配置是否需要重启SQL Server实例以及配置是否为高级选项。

这个技巧演示3个查看SQL Server配置的方法。马上就要讲解的第一个方法显示了基本的选项。第二个方法显示了“高级”选项，或者说在修改之前需要有经验的数据库管理员特别考虑的选项。第三个，也是最后一个方法，展示了如何查询系统目录视图sys.configurations：


```
-- 显示基本的选项
EXEC sp_configure
GO
```

它返回基本配置以及它们的当前值：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

name	minimum	maximum	config_value	run_value
allow updates	0	1	0	0
backup compression default	0	1	0	0
clr enabled	0	1	0	0
cross db ownership chaining	0	1	0	0
default language	0	9999	0	0
filestream access level	0	2	2	2
max text repl size (B)	-1	2147483647	65536	65536
nested triggers	0	1	1	1
remote access	0	1	1	1
remote admin connections	0	1	0	0
remote login timeout (s)	0	2147483647	20	20
remote proc trans	0	1	0	0
remote query timeout (s)	0	2147483647	600	600
server trigger recursion	0	1	1	1
show advanced options	0	1	0	0
user options	0	32767	0	0

下一个查询显示了高级选项（不包含基本选项）：

```
-- 显示高级选项
EXEC sp_configure 'show advanced option', 1
RECONFIGURE
GO

EXEC sp_configure
GO
```

它返回了基本选项和高级选项（没有显示所有行）：

name	minimum	maximum	config_value	run_value
Ad Hoc Distributed Queries	0	1	0	0
affinity I/O mask	-2147483648	2147483647	0	0
affinity mask	-2147483648	2147483647	0	0
Agent XPs	0	1	1	1
allow updates	0	1	0	0
...				
user options	0	32767	0	0
Web Assistant Procedures	0	1	0	0
xp_cmdshell	0	1	0	0

最后，查询视图sys.configurations显示SQL Server的配置，并通过配置名称排序：

```
SELECT name, value, minimum, maximum, value_in_use, is_dynamic, is_advanced
FROM sys.configurations
ORDER BY name
```

这返回所有的选项，除此之外还有其他有用的信息，比如是否为高级选项和是否为动态的。如果选项的is_dynamic值为1，则在执行RECONFIGURE命令之后配置改变就会生效：

name	value	minimum	maximum	value_in_use	is_dynamic	is_advanced
Ad Hoc Distributed Queries	0	0	1	0	1	1
affinity I/O mask	0	-2147483648	2147483647	0	0	1
affinity mask	0	-2147483648	2147483647	0	1	1
affinity64 I/O mask	0	-2147483648	2147483647	0	0	1
affinity64 mask	0	-2147483648	2147483647	0	1	1
...						
Web Assistant Procedures	0	0	1	0	1	1
xp_cmdshell	0	0	1	0	1	1

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在技巧的第一个部分，使用系统存储过程sp_configure返回了基本选项。基本选项的示例包括clr-enabled和nested triggers配置。clr enabled选项表明了在SQL Server实例中是否允许基于CLR的对象。nested triggers配置决定触发其他触发器的触发器是否可以触发。在默认情况下，这些都是所有SQL Server用户可以查看的基本设置。

技巧的第二部分演示了如何查看所有的服务器选项，包括高级选项。为此，必须真正修改SQL Server配置。将show advanced option设置从0（假）配置为1（真）：

```
EXEC sp_configure 'show advanced option', 1
RECONFIGURE
GO
```

在执行完sp_configure之后，使用了RECONFIGURE命令。对于那些不需要重启的SQL Server选项来说，RECONFIGURE命令强制更新当前的配置值。如果使用了不合法或不被推荐的值，RECONFIGURE将会拒绝执行。在大多数情况下，使用RECONFIGURE WITH OVERRIDE将覆盖这个合法性验证。例如，recovery interval选项，它指定了数据库恢复时间的最大值（以分钟为单位）。使用RECONFIGURE设置这个选项的值大于60分钟的话，将引发指示该值不被推荐的警告。虽然这样，这个警告不会阻止你做这个修改。使用RECONFIGURE WITH OVERRIE将强制修改这个选项的值。

在将show advanced option的值改为1后，通过sp_configure返回所有选项：

```
EXEC sp_configure
GO
```

在这个技巧的最后，查询系统目录视图sys.configurations返回所有SQL Server的选项。它返回每个设置的其他信息，包括设置是否为动态的以及是否为高级选项。

改变SQL Server配置

SQL Server很好地实现了开箱即用式的自管理，在大多数情况下，默认的设置就足够了。当必须修改默认配置值时，需要小心地操作，确认你已经完全理解了正在修改的内容。例如，locks配置，它决定了SQL Server可以使用的可用锁的最大值，应该留给SQL Server去管理，允许SQL Server去分配、
617 释放，以及当它需要时提升锁类型。

在这个技巧中，将演示使用sp_configure禁止查询并行性，也包括设置SQL Server实例允许使用的最大内存数量（以MB为单位）。max degree of parallelism选项设置了在并行计划执行中使用的处理器数量限制。这个选项的默认值为使用所有可用的处理器（选项等于0）：

```
SELECT name, value_in_use
FROM sys.configurations
WHERE name IN ('max degree of parallelism')
```

这个查询返回：

```
name                        value_in_use
max degree of parallelism   0
```

在这个示例中，maximum degree of parallelism设置为1个CPU：

```
EXEC sp_configure 'max degree of parallelism', 1
RECONFIGURE
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回：

```
配置选项'max degree of parallelism'已从0更改为1。
请运行RECONFIGURE语句进行安装。
```

现在再一次检查这个值：

```
SELECT name, value_in_use
FROM sys.configurations
WHERE name IN ('max degree of parallelism')
```

这个查询返回：

```
name                        value_in_use
max degree of parallelism   1
```

max server memory选项指定SQL Server允许使用的最大内存数量，以MB为单位。这个设置的默认值不会设为最大值，正如这个查询所展示的：

```
SELECT name, value_in_use
FROM sys.configurations
WHERE name IN ('max server memory (MB)')
```

它返回默认的内存值（非常大的值）：

```
name                      value_in_use
max server memory (MB)    2147483647
```

在这个示例中，SQL Server实例允许使用的最大内存为2500MB：

```
EXEC sp_configure 'max server memory', 2500
RECONFIGURE
GO
```

它返回：

```
配置选项'max server memory(MB)'已从2147483647更改为2500。
请运行RECONFIGURE语句进行安装。
```

然后验证新的值：

```
SELECT name, value_in_use
FROM sys.configurations
WHERE name IN ('max server memory (MB)')
```

这个查询返回：

```
name                    value_in_use
max server memory (MB)  2500
```

解析

在这个技巧中，演示了将max degree of parallelism设置为1，它的意思是在一个查询中只可以使用一个处理器（禁用SQL Server一个查询使用多个CPU的功能）。这个技巧也演示了限制服务器内存最大值为2500MB。只要其他选项没有限制SQL Server的配置，SQL Server将仍然动态管理内存，但仅限于sp_configure指定的最大值。设置的修改也不需要重启SQL Server实例，因此在执行时使用RECONFIGURE命令就足够了。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第22章 创建和配置数据库

在本章中，你将看到各种关于创建和配置SQL Server数据库的技巧。你将学习使用T-SQL来做以下一些工作。

- 创建新的数据库。
- 从数据库添加或删除文件或文件组。
- 查看和修改数据库设置。
- 增加或减少数据库或数据库文件的大小。
- 从SQL Server实例中删除数据库。
- 从SQL Server实例中分离和附加数据库。

本章也将介绍各种“状态”设置，比如配置数据库为只读，或将数据库设置为单用户模式。

22.1 创建、修改和删除数据库

在这第一组技巧中，会讲到如何在SQL Server实例中创建、修改和删除数据库。具体来说，将展示如何做以下工作。

- 创建基于系统数据库model的默认配置的数据库。
- 查看关于数据库的配置信息。
- 使用显式的文件选项创建数据库（而不是依赖系统数据库model）。
- 使用用户定义文件组创建数据库。
- 改变既有数据库的名称。
- 从SQL Server实例中删除数据库。
- 从SQL Server实例中分离数据库，只保留底层数据和日志文件。使用相同的文件附加数据库。

CREATE DATABASE和ALTER DATABASE是用来创建和修改数据库的主要命令。与第1章中讨论的SELECT语句相似，在本章中，每个技巧都会分割成执行特定任务的几个相关部分，不会用大段的内容来讲解语法细节。

22.1.1 使用默认配置创建数据库

这个技巧演示了如何通过使用基于系统数据库model的默认配置，以最简单的形式创建数据库。数据库model是随SQL Server一起安装的系统数据库，并为创建在SQL Server实例中的所有其他数据库定义了模块。如果创建数据库时除了数据库名称以外没有指定任何选项，选项的值将基于系统数据库model。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

创建基于model的数据库的语法如下：

```
CREATE DATABASE database_name
```

CREATE DATABASE命令最简单的形式中只包含一个参数：新数据库的名称。

这个技巧创建了一个名为BookStore的新数据库：

```
USE master
GO

IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'BookStore')

CREATE DATABASE BookStore

GO
```

解析

在这个技巧中，创建了名为BookStore的新数据库，除了数据库名称外没有指定任何其他选项。通过忽略细节（比如文件位置、大小、文件增长选项），创建了基于系统数据库model的新数据库。数据库将包含所有放入model数据库的用户定义对象，并且使用基于新数据库名称的文件命名惯例。

尽管这是创建新数据库的快速方法，它不能让我们对本章中将要描述的一些选项上进行很多控制。

22.1.2 查看数据库信息

这个技巧演示了如何使用系统存储过程sp_helpdb查看数据库的属性以及文件信息：

```
EXEC sp_helpdb 'BookStore'
GO
```

它返回如下的两个结果集（尽管一些内容因打印页面的限制做了一点压缩）：

name	db_size	owner	dbid	created	status	compatibility_level
BookStore	1.62 MB	CAESAR\Administrator	6	Dec 23 2007	Status=ONLINE, Updateability=READ_WRITE, UserAccess=MULTI_USER, Recovery=FULL, Version=639, Collation=SQL_Latin1_General_CP1_CI_AS, SQLSortOrder=52, IsAutoCreateStatistics, IsAutoUpdateStatistics, IsFullTextEnabled	100

name	fileid	filename	filegroup	size	maxsize	growth	usage
BookStore	1	C:\Program Files\Microsoft SQL Server\MSSQL10.AUGUSTUS\MSSQL\DATA\BookStore.mdf	PRIMARY	1152 KB	Unlimited	1024 KB	data only
BookStore_log	2	C:\Program Files\Microsoft SQL Server\MSSQL10.AUGUSTUS\MSSQL\DATA\BookStore_log.LDF	NULL	504 KB	2147483648 KB	10%	log only

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

使用系统存储过程sp_helpdb查看数据库的属性。这个系统存储过程只有一个可选的参数，在这个例子中是数据库的名称：

```
EXEC sp_helpdb 'BookStore'
```

如果在这个存储过程中忽略数据库的名称，将返回SQL Server实例中所有数据库的信息。

这个系统存储过程返回的信息如下所示。

- 数据库的名称和拥有者。
- 创建数据库的日期。
- 各种数据库设置和选项，比如数据库默认的排序规则或是否自动更新统计信息（数据库选项会在本章稍后描述）。
- 组成数据库的各个文件列表，包括文件大小、文件组和文件增长选项。

输出也包括数据库的兼容级别。例如，SQL Server 2008数据库默认拥有的兼容级别为100。SQL Server 2005默认的兼容级别为50，SQL Server 2000默认的兼容级别为80。兼容级别允许在SQL Server 2008中的数据库与SQL Server以前的版本兼容。这也意味着不可以在SQL Server 2005兼容的数据库中使用SQL Server 2008中引入的T-SQL扩展。在以前的SQL Server版本中，可以使用系统存储过程sp_dbcmptlevel设置这个级别。在SQL Server 2008中，使用ALTER DATABASE命令设置数据库兼容级别，例如：

623

```
ALTER DATABASE AdventureWorks
SET COMPATIBILITY_LEVEL = 100
GO
```

22.1.3　使用文件选项创建数据库

如果只是指望创建应急的测试数据库，则使用系统数据库model的默认选项创建新数据库是可以的，但是在正式环境下通常会希望更多地考虑数据库数据和日志文件的位置、大小和增长选项。这个技巧将演示创建新数据库时指定显式的文件选项的使用。

CREATE DATABASE的部分语法如下所示：

```
CREATE DATABASE database_name
   [ ON
      [ <filespec> [ ,...n ] ] ]
[ [ LOG ON { <filespec> [ ,...n ] } ] ]
```

表22-1简要地描述了这个语法的参数。

表22-1　CREATE DATABASE的文件参数

参　数	描　述
database_name	定义数据库的名称
[ON [<filespec> [,...n]]]	为数据库指定一个或多个显式定义的数据文件
[LOG ON { <filespec> [,...n] }]	为数据库指定一个或多个显式定义的事务日志文件

创建数据文件和日志文件的参数filespec的语法如下：

```
[ PRIMARY ]
(
   [ NAME = logical_file_name , ]
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    FILENAME = 'os_file_name'
       [ , SIZE = size [ KB | MB | GB | TB ] ]
       [ , MAXSIZE = { max_size [ KB | MB | GB | TB ] | UNLIMITED } ]
       [ , FILEGROWTH = growth_increment [ KB | MB | % ] ]
) [ ,...n ]
```

表22-2描述了参数filespec。

表22-2 filespec参数

参 数	描 述
PRIMARY	这个可选的关键字指定filespec中的数据文件为主数据文件(数据库的入口，它包含指向其他文件的指针，并且通常用扩展名.mdf来命名)。一个数据库只允许拥有一个主文件，并且如果没有显式指定它，那么CREATE DATABASE中列出的第一个数据库文件将用作主文件
logical_file_name	这个参数定义了数据库文件的逻辑名称
os_file_name	这个参数指定了数据库文件的物理路径和文件名称
size [KB \| MB \| GB \| TB]	这个参数定义了文件的初始大小，基于选择的大小属性（千字节、兆字节、千兆字节、兆兆字节）
MAXSIZE = { max_size [KB \| MB \| GB \| TB] \| UNLIMITED }	这个参数指定了文件的最大允许大小。如果指定UNLIMITED，文件会持续增长到物理磁盘空间已满
FILEGROWTH = growth_increment [KB \| MB \| %]	这个参数表示当需要空间时文件大小的增量。可以指定千字节或兆字节，或是既有文件大小要增长的百分比。如果指定0，文件将不会增长
[,...n]	这个参数表示你可以指定一个或多个定义的文件（每个数据库最多32 767个文件）

在这个技巧中，将使用如上所述的CREATE DATABASE选项来创建名为BookStoreArchive的新数据库：

```
USE master
GO

CREATE DATABASE BookStoreArchive
ON PRIMARY
(  NAME = 'BookStoreArchive',
   FILENAME = 'F:\Apress\BookStoreArchive.mdf' ,
   SIZE = 3MB ,
   MAXSIZE = UNLIMITED,
   FILEGROWTH = 10MB ),
(  NAME = 'BookStoreArchive2',
   FILENAME = 'G:\Apress\BookStoreArchive2.ndf' ,
   SIZE = 1MB ,
   MAXSIZE = 30,
   FILEGROWTH = 5% )
   LOG ON
(  NAME = 'BookStoreArchive_log',
   FILENAME = 'H:\Apress\BookStoreArchive_log.LDF' ,
   SIZE = 504KB ,
   MAXSIZE = 100MB ,
   FILEGROWTH = 10%)
GO
```

解析

在这个技巧中，创建了名为BookStoreArchive的数据库。使用关键字PRIMARY来指定第一个文件为主数据文件：

624

22

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE DATABASE BookStoreArchive
ON PRIMARY
```

第一个文件定义放在后面的括号中。逻辑文件名为BookStoreArchive：

625

```
(  NAME = 'BookStoreArchive',
```

物理文件名被指定到F:盘。在实际场景中，你很可能将数据文件放置到不同的盘符下（它支持RAID 5或RAID 10阵列）：

```
FILENAME = 'F:\Apress\BookStoreArchive.mdf' ,
```

下一步，初始的数据文件大小设置为3MB：

```
SIZE = 3MB ,
```

文件的最大设置为UNLIMITED（无限制），意思是它可以持续增长到C:盘已满：

```
MAXSIZE = UNLIMITED,
```

将增长量设置为10MB文件块，任何时候文件需要更多空间时，文件的大小都将增长10MB：

```
FILEGROWTH = 10MB ),
```

前面是以逗号结尾的文件定义，随后是另一个数据文件定义：

```
(  NAME = 'BookStoreArchive2',
  FILENAME = 'G:\Apress\BookStoreArchive2.ndf' ,
```

第二个数据文件指定了不同的逻辑名和物理文件名。物理文件名以.ndf文件扩展名结尾。尽管这个特定的文件扩展名不是必需的，但是如果主文件使用.mdf，所有辅助数据文件使用.ndf，这会让文件类型更容易识别。假定所有盘符在不同的通道或控制器上使用了RAID，那么将多个文件分散添加到不同的盘符下，可以分离I/O活动，并且潜在地提升更大的高业务量数据库的性能。

第二个文件的大小设置为了1MB，并且限定最大文件为30MB。文件增长设置为5%，而不是像定义第一个数据文件时那样使用MB：

```
SIZE = 1MB ,
MAXSIZE = 30,
FILEGROWTH = 5% )
```

在定义完两个数据文件之后，关键字LOG ON标志了事务日志文件定义的开始：

```
LOG ON
(  NAME = 'BookStoreArchive_log',
```

物理文件名使用了.ldf文件扩展名，这是事务日志文件的标准：

```
FILENAME = 'H:\Apress\BookStoreArchive_log.LDF' ,
```

初始大小设置为504KB，最大的事务日志大小为100MB，并具有10%的文件增长率。

```
SIZE = 504KB ,
MAXSIZE = 100MB ,
FILEGROWTH = 10%)
```

一旦执行了CREATE DATABASE命令，服务器上就会自动创建相关的文件，然后数据库就可以使用了。

在本章稍后的技巧中，会讲解如何修改既有文件属性，以及如何将新数据或事务文件添加到数据

626 库。

22.1.4　使用用户定义文件组创建数据库

一个数据库至少必须拥有一个数据文件和一个事务日志文件。这些文件属于一个数据库，并且不

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

可以被其他数据库共享。默认情况下，当创建数据库时，数据文件将属于主文件组。文件组（filegroup）是为了便于管理和分配而命名的集合起来的文件。主文件组包含主数据文件，以及其他没有显式分配到不同文件组的数据文件。数据文件（不是事务日志文件）属于文件组。

除了主文件组之外（这是所有SQL Server数据库所拥有的），还可以创建用来放置文件的辅助的用户定义文件组。用户定义文件组经常使用在超大型数据库（VLDB）中，允许在几个阵列中对数据库进行分区，并在文件组的级别管理备份，而不是在整个数据库的级别。

注解 可以在指定的文件组上放置表或索引。关于介绍文件组和表的内容请看第4章，介绍文件组和索引的内容请看第5章。

在这个技巧中，演示了如何在用户定义文件组上创建具有文件的数据库。完成这个操作的语法如下：

```
CREATE DATABASE database_name
[ ON
FILEGROUP filegroup_name [ CONTAINS FILESTREAM ] [ DEFAULT ]
    <filespec> [ ,...n ]
]
[
    [ LOG ON { <filespec> [ ,...n ] } ] ]
```

表22-3详细描述了语法的参数。

表22-3 CREATE DATABASE的参数

参 数	描 述
database_name	定义了数据库的名称
FILEGROUP filegroup_name [CONTAINS FILESTREAM DEFAULT]	指定文件组的逻辑名称。如果跟随了关键字DEFAULT，这个文件组将作为数据库的默认文件组（意思是默认的情况下所有对象都将创建到这里）
<filespec> [,...n]	为数据库指定一个或多个显式定义的数据文件
[LOG ON { <filespec> [,...n] }]	为数据库指定一个或多个显式定义的事务日志文件

这个技巧创建了名为BookStoreInternational的新数据库，它使用了两个文件组。一个是必需的主文件组，另一个新的用户定义文件组FG2在CREATE DATABASE命令中创建：

```
USE master
GO

CREATE DATABASE BookStoreInternational
ON PRIMARY
(  NAME = 'BookStoreInternational',
  FILENAME = 'C:\Apress\BookStoreInternational.mdf',
  SIZE = 3MB ,
  MAXSIZE = UNLIMITED,
  FILEGROWTH = 5MB ),
FILEGROUP FG2 DEFAULT
(  NAME = 'BookStoreInternational2',
  FILENAME = 'C:\Apress\BookStoreInternational2.ndf',
  SIZE = 1MB ,
  MAXSIZE = UNLIMITED,
  FILEGROWTH = 1MB )
LOG ON
(  NAME = 'BookStoreInternational_log',
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
  FILENAME = 'C:\Apress\BookStoreInternational.ldf',
  SIZE = 504KB ,
  MAXSIZE = 100MB ,
  FILEGROWTH = 10%)
GO
```

解析

在这个技巧中，创建了具有两个数据文件和一个事务日志文件的新数据库。第一个数据文件创建在PRIMARY文件组中，使用关键字FILEGROUP，将第二个数据库数据文件创建在新的名为FG2的用户定义文件组中。这个文件组标记为默认的文件组，因此在数据库中创建的任何新数据库对象都将创建在这个文件组中：

```
...
FILEGROUP FG2 DEFAULT
(  NAME = 'BookStoreInternational2',
  FILENAME = 'C:\Apress\BookStoreInternational2.ndf',
  SIZE = 1MB ,
  MAXSIZE = UNLIMITED,
  FILEGROWTH = 1MB )
...
```

由于事务日志不会放置到文件组中，因此在标准filespec定义中使用关键字LOG ON。

在这个技巧中，在文件组FG2中放置了一个文件，当然也可以在一个文件组中放置多个文件。如果在一个文件组中放置多个文件，SQL Server将按比例填充每一个文件，而不是填充一个文件后再填充下一个。

22.1.5　设置数据库用户访问

SQL Server提供3种数据库用户访问模式，它们决定哪些（以及多少）用户可以访问数据库：SINGLE_USER、RESTRICTED_USER和MULTI_USER。SINGLE_USER和RESTRICTED_USER选项是对在数据库中执行活动的其他用户“关闭大门”的方法。在需要执行不允许其他用户在同一时间在数据库中活动的数据库配置修改操作时，这经常是有用的。当需要撤销数据修改，或者在切换到新系统或应用程序升级之前强制用户退出时，也可以使用这些选项。随后的表格更加详细地描述了所有选项。

628

修改用户访问的部分语法如下：

```
ALTER DATABASE database_name
SET { SINGLE_USER | RESTRICTED_USER | MULTI_USER }
[WITH { ROLLBACK AFTER integer [ SECONDS ]
  | ROLLBACK IMMEDIATE
  | NO_WAIT
}]
```

表22-4描述了这个语法的参数。

表22-4　ALTER DATABASE的参数

参　数	描　述
database_name	这个参数定义了要修改用户访问的既有数据库的名称
SINGLE_USER \| RESTRICTED_USER \| MULTI_USER	当指定SINGLE_USER时，同一时间只允许一个用户访问数据库。当指定这个选项时，除非使用终止选项（终止选项请看表格中下一行的描述），否则会封锁修改，直到所有其他用户从数据库中断开连接。使用RESTRICTED_USER选项时，只有sysadmin、dbcreator或db_owner角色的成员可以访问数据库。使用MULTI_USER时，所有对数据库有权限的用户都允许访问

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

参 数	描 述
ROLLBACK AFTER integer [SECONDS] \| ROLLBACK IMMEDIATE \| NO_WAIT	这些终止选项允许回滚在ALTER DATABASE语句执行时数据库中未完成的事务。如果没有使用终止选项，则ALTER语句可能不得不等待，直到锁定的连接完成任务。实际上它们也可以用在SET子句中，但是当将数据库变为SINGLE_USER或RESTRICTED_USER模式时，使用它们的情况最多。ROLLBACK AFTER integer [SECONDS]指定打开的数据库事务在指定的秒数后回滚。ROLLBACK IMMEDIATE立即回滚打开的事务。当指定NO_WAIT时，如果不能立即完成将引起语句执行失败（为了可以成功执行，使用这个选项需要数据库中没有打开的事务）

这个技巧演示了将AdventureWorks数据库置为SINGLE_USER模式，回滚所有打开的事务，然后将数据库置回到MULTI_USER模式：

```
-- 关闭行计数消息
SET NOCOUNT ON

SELECT user_access_desc
FROM sys.databases
WHERE name = 'AdventureWorks'

ALTER DATABASE AdventureWorks
SET SINGLE_USER
WITH ROLLBACK IMMEDIATE

SELECT user_access_desc
FROM sys.databases
WHERE name = 'AdventureWorks'

ALTER DATABASE AdventureWorks
SET MULTI_USER

SELECT user_access_desc
FROM sys.databases
WHERE name = 'AdventureWorks'
```

这个查询返回：

```
user_access_desc
MULTI_USER

user_access_desc
SINGLE_USER

user_access_desc
MULTI_USER
```

解析

在这个技巧中，查询系统目录视图sys.databases检查当前用户访问模式。然后将数据库改变为SINGLE_USER模式，并终止其他数据库用户会话中所有打开的事务：

```
ALTER DATABASE AdventureWorks
SET SINGLE_USER
WITH ROLLBACK IMMEDIATE
```

然后通过sys.databases检查用户访问模式，并且将数据库改回MULTI_USER模式：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ALTER DATABASE AdventureWorks
SET MULTI_USER
```

在这之后，再一次通过sys.databases检查访问模式：

```
SELECT user_access_desc
FROM sys.databases
WHERE name = 'AdventureWorks'
```

根据应用程序处理未完成的进程的方法，以这种方式取消打开的事务可能会在应用程序中引发问题，要记住这个重要的问题。要尽可能在不活动或没有事务活动的期间尝试改变用户访问模式。但是对于某些操作你需要将数据库设置为SINGLE_USER，比如说READ_ONLY和READ_WRITE选项。要关闭所有当前用户连接的另外一个原因可能是，例如，在不用处理应用程序的阻塞或者错误的情况下进行紧急修复。

630

22.1.6　重命名数据库

在这个技巧中，演示了如何使用ALTER DATABASE命令修改既有数据库的名称。

语法如下：

```
ALTER DATABASE database_name
MODIFY NAME = new_database_name
```

这个命令的两个参数包含了原始数据库名称和新数据库的名称。

这个技巧演示了将BookWarehouse数据库的名称改为数据库BookMart。为了断开到数据库的所有其他并发连接，还要执行ALTER DATABASE...SET SINGLE USER。

```
USE master
GO

-- 创建演示数据库
CREATE DATABASE BookWarehouse
GO

ALTER DATABASE BookWarehouse
SET SINGLE_USER
WITH ROLLBACK IMMEDIATE
GO

ALTER DATABASE BookWarehouse
MODIFY NAME = BookMart
GO

ALTER DATABASE BookMart
SET MULTI_USER
GO
```

这个查询返回（基于在终止连接操作期间的数据库中的活动，结果可能会不同）：

```
正在回滚不合法事务。估计回滚已完成：100%。
数据库名称'BookMart' 已设置。
```

解析

在这个技巧中，数据库从BookWarehouse重命名为BookMart。在执行这个操作之前，查询会话的上下文改变到数据库master（因为你不可以使用到数据库本身的连接改变数据库的名称）：

```
USE master
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

先创建用于演示的新数据库。将新数据库设置为单用户模式，并且回滚数据库内的所有活动事务（除了当前会话中存在的事务）：

```
ALTER DATABASE BookWarehouse
SET SINGLE_USER
WITH ROLLBACK IMMEDIATE
GO
```

631

然后使用ALTER DATABASE和MODIFY NAME命令修改数据库名称：

```
ALTER DATABASE BookWarehouse
MODIFY NAME = BookMart
GO
```

尽管数据库在原始名称下被设置为单用户模式，在它被显式设置为MULTI_USER访问模式之前，它将一直保持单用户的访问模式。

```
ALTER DATABASE BookMart
SET MULTI_USER
GO
```

22.1.7 删除数据库

可以使用DROP DATABASE命令从SQL Server中删除一个用户数据库。DROP DATABASE从SQL Server系统表中删除对数据库的引用。如果其中的文件是联机的，它也将从SQL Server服务器中删除物理文件。

语法如下：

```
DROP DATABASE database_name
```

22

在这个技巧中，删除数据库BookStoreArchive_Ukrainian：

```
USE master
GO

-- 创建演示数据库
CREATE DATABASE BookStoreArchive_Ukrainian
GO

ALTER DATABASE BookStoreArchive_Ukrainian
SET SINGLE_USER
WITH ROLLBACK IMMEDIATE
GO

DROP DATABASE BookStoreArchive_Ukrainian
GO
```

解析

在这个技巧中，首先将当前查询会话切换到数据库master，因为当你连接到这个数据库时，是不可以删除它的。这个技巧也会将数据库设置到单用户模式，并强制立即回滚所有打开的事务。最后，使用DROP DATABASE命令删除了数据库。

22.1.8 分离数据库

删除数据库时，同时也从SQL Server实例中删除了它的物理文件。如果希望从SQL Server实例中删除数据库，但仍然保留物理文件（为了归档或将数据库迁移到另一SQL Server实例），你可以使用分离数据库。通过从一个SQL Server实例分离数据库，并在另一个实例添加它，也可以将数据库从一个SQL Server实例移到另一实例。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

632 为了分离数据库，可以使用系统存储过程sp_detach_db，它使用如下的语法：

```
sp_detach_db [ @dbname= ] 'dbname'
    [ , [ @skipchecks= ] 'skipchecks' ]
```

表22-5描述了这个存储过程的参数。

表22-5　sp_detach_db参数

参　数	描　述
dbname	这个选项提供要分离的数据库名称
skipchecks	这个选项可以取true或false值。当这个选项为true时，在分离数据库之前不会更新统计信息。默认情况下更新统计信息

注意　这个系统存储过程还有一个参数@keepfulltextindexfile，我没有把它包含在这里。由于全文索引元数据现在已经存在于数据库中，SQL Server今后的版本将会删除它。

在这个技巧中，将创建一个数据库，然后使用sp_detach_db分离这个数据库：

```
-- 创建要分离的默认的数据库
USE master
GO

CREATE DATABASE TestDetach
GO

-- 踢掉当前数据库中所有用户

ALTER DATABASE TestDetach
SET SINGLE_USER
WITH ROLLBACK IMMEDIATE

-- 分离数据库

EXEC sp_detach_db 'TestDetach',
        'false' -- 不忽略检查
```

这个查询返回如下部分结果：

```
正在更新[sys].[queue_messages_1977058079]
    [queue_clustered_index]，不需要更新...
    [queue_secondary_index]，不需要更新...
    已更新0 条索引/统计信息，不需要更新。
...

正在更新[sys].[ syscommittab]
    [ci_commit_ts]，不需要更新...
    [si_xdes_id]，不需要更新...
    已更新0 条索引/统计信息，不需要更新。

633 已更新了所有表的统计信息。
```

解析

在这个技巧中，创建了一个名为TestDetach的新数据库，在这之后，使用ALTER DATABASE命令设置数据库TestDetach为单用户模式，并使用ROLLBACK IMMEDIATE选项踢掉所有打开的数据库连接。

然后使用系统存储过程sp_detach_db分离数据库——但是要在更新了统计信息（在第二个参数中

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

指定false）之后分离。现在，对所有的意图和目的来说，数据库都已经删除了。但数据文件仍然存在于SQL Server实例的服务器中，并且可以在当前或其他SQL Server实例中重新创建（如果你想这样做）。

22.1.9 附加数据库

前面的技巧演示了如何分离数据库。在下面的技巧中，将演示如何附加数据库。假定数据库的副本不需要分别保留在两个SQL Server实例中，那么使用分离/附加的方式将数据库从一个SQL Server实例移到另一个实例是个很好的方法。

警告 从一个服务器到另一个服务器分离和附加数据库并不会移动关联到数据库用户的SQL Server登录名。必须执行独立的操作来将登录名移动到新的SQL Server实例中。

使用CREATE DATABASE FOR ATTACH命令将数据库附加到SQL Server实例中。

部分语法如下：

```
CREATE DATABASE database_name
    ON <filespec> [ ,...n ]
    FOR { ATTACH
        | ATTACH_REBUILD_LOG }
```

表22-6描述了这个命令的参数。

表22-6 CREATE DATABASE...FOR ATTACH

参 数	描 述
database_name	要附加的数据库名称
<filespec> [,...n]	主数据文件和所有其他数据库文件的名称。如果原始的分离的数据库的文件位置与既有的文件位置匹配，则只需要引用主数据文件。但是如果文件位置已经发生变化，则应该为每个数据库文件指定位置
ATTACH \| ATTACH_REBUILD_LOG	ATTACH指定使用在分离的数据库中使用的所有原始文件来创建数据库。当指定ATTACH_REBUILD_LOG，并且事务日志文件不可用时，SQL Server将重建事务日志文件

在这个技巧中，在前一个技巧中分离的数据库TestDetach现在将使用相同的文件和文件路径重新附加到SQL Server实例中。但是，数据库将使用新的名称TestAttach重新附加到SQL Server实例中：

```
CREATE DATABASE TestAttach
    ON (FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL10.AUGUSTUS
\MSSQL\DATA\TestDetach.mdf')
    FOR ATTACH
```

解析

在这个技巧中，使用CREATE DATABASE FOR ATTACH命令重新附加数据库。这个命令引用了主数据文件名称，它包含了对其他文件位置的引用（在这个例子中是事务日志文件）。

如果分离了数据库，然后重新定位了辅助数据文件和/或事务日志文件，你将需要显式地引用在CREATE DATABASE...FOR ATTACH命令中每个文件的新位置。在filespec中指定文件的新路径。如果事务日志变为不可用，可以使用ATTACH_REBUILD_LOG而不是使用ATTACH来重建事务日志文件。

22.2 配置数据库选项

在下面的一组技巧中会讲到如何配置影响数据库中活动的行为的数据库选项。后面部分将具体讲

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解如何完成以下操作。

- 查看当前数据库配置的数据库选项。
- 配置ANSI SQL 选项。
- 配置自动选项。自动数据库选项影响SQL Server数据库引擎的行为、启用或禁用自动维护或元数据更新。
- 配置外部访问选项，包括DB_CHAINING和TRUSTWORTHY。
- 使用指定的排序规则创建或修改数据库。
- 配置游标选项。
- 启用日期相关性优化。启用DATE_CORRELATION_OPTIMIZATION选项可以优化通过datetime外键引用相关联的两个表。
- 修改数据库参数化行为。通过ALTER DATABASE的PARAMETERIZATION选项控制是所有的还是部分数据库的查询参数化。
- 启用行版本。SQL Server有两个允许语句级别和事务级别读取一致性的数据库选项：ALLOW_SNAPSHOT_ISOLATION和READ_COMMITTED_SNAPSHOT。
- 配置数据库恢复模式。SQL Server使用3种不同的恢复模式，它们定义了是否可以进行事务日志备份，以及在可以的情况下何种数据库活动将写入事务日志。
- 配置页验证。SQL Server有3种模式来控制和检测磁盘错误引发的未完成的I/O事务：CHECKSUM、TORN_PAGE_DETECTION和NONE。

注意　一些数据库选项已经在其他章节演示了。关于ENCRYPTION数据库选项的讨论，请看第19章。关于AUTO_CLEANUP和CHANGE_RETENTION数据库选项的讨论，请看第2章。Service Broker数据库选项的讨论请看第20章。

635

首先介绍如何通过使用系统目录视图sys.databases来查看数据库的当前数据库选项。

22.2.1　查看数据库选项

这个技巧演示了如何使用系统目录视图sys.databases查看AdventureWorks数据库的数据库选项：

```
SELECT name, is_read_only, is_auto_close_on, is_auto_shrink_on
FROM sys.databases
WHERE name = 'AdventureWorks'
```

这个查询返回：

```
name              is_read_only    is_auto_close_on    is_auto_shrink_on
AdventureWorks    0               0                   0
```

解析

在这个技巧中，使用查询来查看3个数据库选项：is_read_only、is_auto_close_on和is_auto_shrink_on。系统目录视图sys.databases还可以用来查看很多其他的数据库选项，包括用户数据库和系统数据库的数据库选项。

22.2.2　配置 ANSI SQL 选项

这个技巧演示了如何设置数据库的ANSI（美国国家标准学会）SQL相容性默认值。表22-7详细介

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

绍了ANSI SQL选项。

表22-7 ANSI SQL选项

选 项	描 述
ANSI_NULL_DEFAULT	当设为ON时，在CREATE或ALTER表语句中未显式定义NULL或NOT NULL的列将默认允许NULL值。默认为OFF，意思是如果未显式定义，列将被定义为NOT NULL
ANSI_NULLS	启用它时，与NULL值的比较将返回UNKNOWN。这个设置的默认选项为OFF，意思是当两个值都为NULL时比较结果为TRUE
ANSI_PADDING	在插入到varchar或nvarchar数据类型列之前，这个选项将字符串填充到相同的长度。默认设置是OFF，意思是字符串将不会被填充
ANSI_WARNINGS	这个设置会影响一些不同的行为。当指定为ON时，在聚合函数中使用任何空值都会引发警告消息。同时，被零除和算术溢出错误将回滚语句并且返回一个错误消息。这个设置默认为OFF
ARITHABORT	当这个选项指定为ON时，溢出或被零除的查询将终止查询并且返回错误。如果这发生在事务中，事务将被回滚。当这个选项指定为OFF（默认的）时，会引发警告，但是语句仍然会继续执行
CONCAT_NULL_YIELDS_NULL	当这个选项设置为ON时，字符串与空值连接将会产生NULL值。当为OFF时（默认的），空值会被视为空字符串来进行处理
NUMERIC_ROUNDABORT	当这个选项指定为ON时，当表达式中发生精度损失时将引发错误。当为OFF时（默认的），不会引发错误消息，但是结果会根据目标列或变量的精度进行舍入
QUOTED_IDENTIFIER	当这个选项指定为ON时，标识符可以使用双引号分隔，并且文字使用单引号分隔。当指定为OFF（默认的）时，只有文字可以使用单引号或双引号分隔
RECURSIVE_TRIGGERS	当这个选项为ON时，可以递归触发触发器（触发器1触发触发器2，它又触发触发器1）。当为OFF时（默认的），不允许递归触发触发器

设置这些选项的语法如下：

```
ALTER DATABASE database_name
SET  <option> { ON | OFF }
```

这个语句使用了两个参数：要修改的数据库名称，以及希望启用或禁用的ANSI SQL设置的名称。

注解 所有新创建的数据库的默认选项将基于创建新数据库时数据库model中的值。但是在这个语法中强调的SQL Server的默认值是“开箱即用”的。

在这个技巧中，使用ALTER DATABASE命令设置ANSI_NULLS选项为OFF。这意味着查询中与NULL值进行比较，并且两个值都为NULL时，结果将视为TRUE：

```
SET NOCOUNT ON

SELECT is_ansi_nulls_on
FROM sys.databases
WHERE name = 'AdventureWorks'

ALTER DATABASE AdventureWorks
SET ANSI_NULLS OFF

SELECT is_ansi_nulls_on
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FROM sys.databases
WHERE name = 'AdventureWorks'
```

这个查询返回：

```
is_ansi_nulls_on
1

is_ansi_nulls_on
0
```

637

解析

这个技巧演示了使用ALTER DATABASE命令修改ANSI SQL设置。技巧首先通过查询系统目录视图sys.databases来查看当前数据库设置。在这之后，使用ALTER TABLE和SET ANSI_NULLS OFF将ANSI_NULLS设置转换为OFF，然后再一次查询系统目录视图sys.databases来确认所做的修改。

必须注意的是，数据库的ANSI选项仍然可以被SET语句连接级别的设置覆盖。例如，尽管AdventureWorks数据库的ANSI_NULLS设置为OFF，在查询中使用SET ANSI_NULLS ON批处理将覆盖查询会话的数据库设置行为。

还有，这里介绍的一些选项需要在计算列或索引视图之上操作索引之前设置为ON。这些选项包括ARITHABORT、QUOTED_IDENTIFIER、CONCAT_NULL_YIELDS_NULL、ANSI_NULLS、ANSI_WARNINGS和ANSI_PADDING。但是，NUMERIC_ROUNDABORT必须为OFF。

22.2.3　配置自动选项

自动数据库选项影响SQL Server数据库引擎的行为，启用或禁用自动维护或元数据更新。表22-8描述了所有的自动选项。

表22-8　自动选项

参　数	描　述
AUTO_CLOSE	启用这个选项时，当数据库的最后一个用户连接存在并且所有操作都已完成时，将关闭数据库并且释放资源
AUTO_CREATE_STATISTICS	当启用这个选项时，SQL Server自动地生成关于列中值分布的统计信息。这个信息帮助查询分析器生成一个可接受的查询执行计划（用来返回查询请求的结果集的内部计划）
AUTO_SHRINK	当启用这个选项时，SQL Server自动地收缩数据和日志文件。只有在文件的未使用空间超过25%时才引发收缩操作。然后收缩数据库至拥有25%的剩余空间，或者是数据或日志文件的原始大小。例如，如果定义的主数据文件为100MB，收缩操作将不能将文件减小到100MB以下
AUTO_UPDATE_STATISTICS	当启用这个选项时，这个选项将自动更新已经为表创建的统计信息
AUTO_UPDATE_STATISTICS_ASYNC	当这个选项为ON时，如果查询启动过期统计信息的自动更新，则该查询在编译前不会等待统计信息的更新。当为OFF时（默认的），启动自动更新过期统计信息的查询将一直等待，直到更新完成才编译查询计划

配置自动数据库选项的语法如下：

```
ALTER DATABASE database_name
SET AUTO_CLOSE { ON | OFF }
```

638

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
| AUTO_CREATE_STATISTICS { ON | OFF }
| AUTO_SHRINK { ON | OFF }
| AUTO_UPDATE_STATISTICS { ON | OFF }
| AUTO_UPDATE_STATISTICS_ASYNC { ON | OFF }
```

第一个参数是要修改的数据库名称。第二个参数是希望启用（ON）或禁用（OFF）的选项的名称。这个技巧将演示为AdventureWorks数据库启用AUTO_UPDATE_STATISTICS_ASYNC自动数据库选项：

```
SET NOCOUNT ON

SELECT is_auto_update_stats_async_on
FROM sys.databases
WHERE name = 'AdventureWorks'

ALTER DATABASE AdventureWorks
SET AUTO_UPDATE_STATISTICS_ASYNC ON

SELECT is_auto_update_stats_async_on
FROM sys.databases
WHERE name = 'AdventureWorks'
```

它返回：

```
is_auto_update_stats_async_on
0

is_auto_update_stats_async_on
1
```

解析

这个技巧演示了使用ALTER DATABASE命令修改AUTO_UPDATE_STATS_ASYNC自动数据库设置。技巧首先通过查询系统目录视图sys.databases来查看当前数据库设置。在这之后，使用ALTER DATABASE将AUTO_UPDATE_STATS_ASYNC设置为ON，然后再一次查询系统目录视图sys.databases来确认所做的修改。

一些自动设置设为ON时会对性能产生负面影响，比如AUTO_CLOSE和AUTO_SHRINK选项。对于AUTO_CLOSE选项来说，存在几分钟没有用户登录的高流量的数据库，在完全关闭数据库后打开数据库的开销会引发性能问题（重复的开启和关闭数据库的开销）。对于AUTO_SHRINK选项来说，SQL Server可能会在不合适的时候启动数据库收缩操作，降低常规最终用户的查询性能。而且，在这个选项设为ON时，数据库大小可能会重复地扩大或缩小。AUTO_SHRINK还会造成碎片，因此允许在索引重建或重组织之后启用这个选项会破坏维护这些活动带来的好处。只要有可能，保持数据库中的剩余空间，这样SQL Server就不会连续地扩大和缩小相同的文件。

除非有非常好的原因，否则就不要将其他选项设置为OFF，包括AUTO_CREATE_STATISTICS和AUTO_UPDATE_STATISTICS选项。统计信息有助于SQL Server编译最佳的查询优化计划，自动创建和维护统计信息的开销和它们提供的查询性能相比通常不重要。

639

22.2.4 创建或修改允许外部访问的数据库

CREATE DATABASE命令提供两个外部访问数据库的选项：DB_CHAINING和TRUSTWORTHY。默认情况下这两个选项都为OFF。当启用DB_CHAINING选项时，新的数据库可以参与到跨数据库所有权链中。最简单的形式，当一个对象（比如视图或存储过程）引用了另一个对象时将引发所有权链。如果包含这些对象架构的拥有者与引用对象的拥有者相同，将不检查引用对象的权限。跨数据库链（cross-database chaining）意思是一个对象在不同的数据库引用另一个对象。所有权链能导致不正确的或者预期之外

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的数据访问——例如，如果启用了DB_CHAINING选项，视图中dbo拥有的架构引用不同数据库的dbo拥有的数据表，将不会检查安全对象。

使用TRUSTWORTHY选项指定SQL Server是否“相信”给定的数据库中的模块或程序集。当这个选项为OFF时，SQL Server将防止数据库的程序集中某些恶意的EXTERNAL_ACCES或UNSAFE活动，或者防止恶意的代码在高特权用户上下文执行。

使用启用或禁用的外部访问选项来创建数据库的部分语法如下：

```
CREATE DATABASE database_name
...
   [ WITH { DB_CHAINING { ON | OFF }
          |TRUSTWORTHY { ON | OFF }}]
```

在事务日志LOG ON选项之后，在WITH子句中有两个选项。它们以相同的语句来启用，且默认情况下它们都为OFF。

也可以使用ALTER DATABASE来对既有的数据库设置这些选项：

```
ALTER DATABASE database_name
{SET DB_CHAINING { ON | OFF }
| TRUSTWORTHY { ON | OFF }}
```

这个技巧演示了如何创建启用数据库链选项的数据库，然后修改新数据库以允许外部数据库访问数据库对象：

```
USE master
GO
-- 使用数据库model的默认值创建数据库
CREATE DATABASE BookData
WITH DB_CHAINING ON
GO

USE master
GO
--现在修改新数据库，将TRUSTWORTHY选项置为ON
ALTER DATABASE BookData
SET TRUSTWORTHY ON
GO
```

解析

在这个技巧中，在CREATE DATABASE语句中启用数据库所有权链。使用基于系统数据model的默认选项（文件名称、大小、增长）创建数据库BookData。在这之后，在WITH子句中启用了数据库所有权链：

640

```
WITH DB_CHAINING ON
```

在这之后，使用ALTER DATABASE命令启用TRUSTWORTHY设置。使用了关键字SET，而不是用WITH，随后是外部访问选项的名称和关键字ON：

```
ALTER DATABASE BookData
SET TRUSTWORTHY ON
GO
```

22.2.5　使用非服务器默认排序规则创建或改变数据库

在这个技巧中，演示了如何使用指定排序规则来创建或修改数据库。SQL Server排序规则决定数据如何进行排序、比较、呈现和存储。某些时候你可能希望以与SQL Server实例默认值不同的代码页或排列顺序存储数据，这时可以使用与安装SQL Server实例时定义的服务器级别排序规则不同的数据

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

库排序规则。

使用CREATE DATABASE指定排序规则的语法如下：

```
CREATE DATABASE database_name
    [ ON
        [ <filespec> [ ,...n ] ]
        [ , <filegroup> [ ,...n ] ] ]
[ [ LOG ON { <filespec> [ ,...n ] } ]
    [ COLLATE collation_name ]]
```

在事务日志定义后使用COLLATE命令显式地定义默认的数据库排序规则。

改变既有数据库的默认排序规则的ALTER DATABASE的语法如下：

```
ALTER DATABASE database_name
{COLLATE collation_name}
```

这个技巧演示了使用Ukrainian排序规则来创建新的数据库，并且使用了不区分大小写和重音设置。在创建了数据库之后，将数据库改变为使用区分大小写和重音的排序规则：

```
CREATE DATABASE BookStoreArchive_Ukrainian
ON PRIMARY
(  NAME = 'BookStoreArchive_UKR',
  FILENAME = 'C:\Apress\BookStoreArchive_UKR.mdf',
  SIZE = 3MB ,
  MAXSIZE = UNLIMITED,
  FILEGROWTH = 10MB )
  LOG ON
(  NAME = 'BookStoreArchive_UKR_log',
  FILENAME = 'C:\Apress\BookStoreArchive_UKR_log.ldf',
  SIZE = 504KB ,
  MAXSIZE = 100MB ,
  FILEGROWTH = 10%)
  COLLATE Ukrainian_CI_AI
  GO

ALTER DATABASE BookStoreArchive_Ukrainian
COLLATE Ukrainian_CS_AS
GO
```

解析

CREATE DATABASE和ALTER DATABASE示例都使用了COLLATE语句，紧随其后的是排序规则的名称，用来指定数据库的默认排序规则。

```
COLLATE Ukrainian_CI_AI
```

在设置了数据库的默认排序规则之后，包含字符数据类型的列（varchar、nvarchar、char、nchar、text、ntext）的新表将使用数据库默认排序规则作为列的排序规则。

警告 使用与SQL Server实例默认值（系统数据库）不同的默认排序规则创建用户定义数据库可能引发排序规则冲突（跨排序规则的数据不可以在查询中转换或连接）。例如，系统数据库tempdb使用了与数据库model相同的排序规则，这可能引发临时表数据操作在连接不同的排序规则数据时失败。一定要彻底地对跨排序规则操作进行测试。

22.2.6 配置游标选项

在第9章我讨论了如何创建和使用T-SQL游标。SQL Server有两个数据库选项来控制T-SQL游标的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

行为，正如你在表22-9中看到的。

表22-9　游标选项

选　项	描　述
CURSOR_CLOSE_ON_COMMIT	当启用CURSOR_CLOSE_ON_COMMIT时，在事务提交后会自动关闭T-SQL游标
CURSOR_DEFAULT { LOCAL \| GLOBAL }	如果启用了CURSOR_DEFAULT LOCAL，则不显式设置GLOBAL作用域创建的游标将默认为本地访问。如果启用了CURSOR_DEFAULT GLOBAL，则不显式设置LOCAL作用域创建的游标将默认为GLOBAL访问

配置游标选项的语法如下：

```
ALTER DATABASE database_name
SET CURSOR_CLOSE_ON_COMMIT { ON | OFF }
  | CURSOR_DEFAULT { LOCAL | GLOBAL }
```

语句使用了两个参数：要修改的数据库名称以及要配置ON或OFF的选项。

这个技巧将演示启用AdventureWorks数据库的CURSOR_CLOSE_ON_COMMIT选项：

```
SET NOCOUNT ON

SELECT is_cursor_close_on_commit_on
FROM sys.databases
WHERE name = 'AdventureWorks'

ALTER DATABASE AdventureWorks
SET CURSOR_CLOSE_ON_COMMIT ON

SELECT is_cursor_close_on_commit_on
FROM sys.databases
WHERE name = 'AdventureWorks'
```

642

这个查询返回：

```
is_cursor_close_on_commit_on
0

is_cursor_close_on_commit_on
1
```

解析

这个技巧演示了使用ALTER DATABASE命令修改CURSOR_CLOSE_ON_COMMIT数据库自动设置。技巧首先通过查询系统目录视图sys.databases来查看当前数据库的设置。在这之后，使用ALTER DATABASE将CURSOR_CLOSE_ON_COMMIT设置转换为ON，然后再次查询系统目录视图sys.databases来确认所做的修改。

22.2.7　启用日期相关性优化

通过引用datetime外键相关的两个表可以从启用DATE_CORRELATION_OPTIMIZATION选项获得益处。当启用它时，SQL Server会收集额外的统计信息，这有助于提升使用两个datetime数据类型列（外键和主键对）的联结的查询性能。

启用这个选项的语法如下：

```
ALTER DATABASE database_name
SET DATE_CORRELATION_OPTIMIZATION { ON | OFF }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个命令使用了两个参数：要修改的数据库名称，以及将DATE_CORRELATION_OPTIMIZATION设置为ON或OFF。这个选项默认为OFF，因为这些表要满足日期相关性优化的准则，将它设为ON会增加额外的开销。

当这个选项为ON时，可以使联结两个表的、通过外键引用相关的datetime值的查询受益。SQL Server将维护额外的相关性统计信息，根据你的查询，它可能会让SQL Server生成更有效率的、更少I/O的密集型查询计划。

为了利用这个数据库设置，并且能自动创建统计信息，至少要有一个datetime列（主键或外键）是聚集索引中的第一个键列或分区表中的分区列。

要知道在更新统计信息时会有额外的开销，可能查询优化带来的好处比不上统计更新带来的开销，因此如果那些主键和外键在datetime关联的表经常更新，那么你应该监视一下数据库性能。

在这个技巧中，AdventureWorks数据库将这个选项设置为ON：

```
SET NOCOUNT ON

SELECT is_date_correlation_on
FROM sys.databases
WHERE name = 'AdventureWorks'
ALTER DATABASE AdventureWorks
SET DATE_CORRELATION_OPTIMIZATION ON

SELECT is_date_correlation_on
FROM sys.databases
WHERE name = 'AdventureWorks'
```

这个查询返回：

```
is_date_correlation_on
0

is_date_correlation_on
1
```

解析

在这个技巧中，使用系统目录视sys.databases来检查AdventureWorks数据库的日期相关性。在这之后，使用了ALTER DATABASE和SET DATE_CORRELATION_OPTIMIZATION ON。最后，再一次检查系统目录视图sys.databases来确认新的设置。

22.2.8 修改数据库参数化行为

在ALTER DATABASE中使用PARAMETERIZATION选项来控制数据库中是所有查询都参数化还是部分查询参数化。

当查询提交到SQL Server时将引发参数化。SQL Server查看SELECT、INSERT、UPDATE和DELETE语句中的文本值，并且尝试对它们进行参数化（生成占位符），因此执行相似的查询时可以重用查询执行计划，而不是为每个查询都生成新的计划。在语句级别对参数化的查询创建执行计划，因此批处理中的每个语句都可以单独进行参数化。

启用这个选项的语法如下：

```
ALTER DATABASE database_name
SET PARAMETERIZATION { SIMPLE | FORCED }
```

这个命令有两个参数：数据库名称和参数化选项。在参数化中有两个选择：SIMPLE（默认的）或FORCED。在SIMPLE参数化的情况下（默认值），SQL语句只对相对较少的查询进行参数化（SQL Server

643

22

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的判断)。设置参数化为FORCED增加了查询参数化的产生，由于创建了更多的查询计划并且可以重用，因此这可以提升查询的性能。

这个技巧演示了如何使用ALTER DATABASE启用这个选项，在sys.databases中检查这个值，然后使用系统目录视图sys.dm_exec_cached_plans和动态管理函数sys.dm_exec_cached_plans显示参数化的结果。首先检查AdventureWorks数据库查看参数化选项是否设置为forced：

```
SELECT is_parameterization_forced
FROM sys.databases
WHERE name = 'AdventureWorks'
```

644 这个查询的结果确认了这个选项没有启用：

```
is_parameterization_forced
0
```

下一步，使用ALTER DATABASE将参数化选项修改为FORCED：

```
ALTER DATABASE AdventureWorks
SET PARAMETERIZATION FORCED
```

通过查询sys.databases来确认所做的修改：

```
SELECT is_parameterization_forced
FROM sys.databases
WHERE name = 'AdventureWorks'
```

它返回：

```
is_parameterization_forced

1
```

然后，为了示范FORCED选项的使用情况，使用DBCC FREEPROCCACHE命令清空过程缓存：

```
-- 警告！不要在运行环境下的SQL Server实例中运行它
-- 它将清空过程缓存，并且引发所有查询重编译
DBCC FREEPROCCACHE
```

然后执行随后的查询：

```
-- 警告！不要在运行环境下的SQL Server实例中运行它
-- 它将清空过程缓存，并且引发所有查询重编译
DBCC FREEPROCCACHE

USE AdventureWorks
GO

SELECT BirthDate
FROM HumanResources.Employee
WHERE BusinessEntityID IN
   (SELECT TOP 3 BusinessEntityID
    FROM Sales.SalesPersonQuotaHistory
    WHERE SalesQuota = 263000.00)
```

这个查询返回：

```
BirthDate
1941-11-17 00:00:00.000
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

现在查询系统目录视图sys.dm_exec_cached_plans。这个视图返回关于SQL Server实例中缓存的查询执行计划的信息。视图的plan_handle列包含了内存中引用查询计划的标识符。为了在内存中查看这个计划，查询使用了动态管理函数sys.dm_exec_query_plan，它将plan_handle作为一个参数并且以XML格式返回执行计划。这个查询从执行过的查询中查找所有引用了EmployeeID、预编译的、缓存的计划： 645

```
SELECT query_plan
FROM sys.dm_exec_cached_plans p
CROSS APPLY sys.dm_exec_query_plan(p.plan_handle)
WHERE CAST(query_plan as varchar(max))
   LIKE '%TOP 3 BusinessEntityID%' AND
   objtype = 'Prepared'
```

它返回如下部分结果（在这里显示一小部分XML格式的计划）：

```
<StmtSimple StatementText="(@0 numeric(38,2))select BirthDate from HumanResources .
 Employee
where BusinessEntityID in ( select top 3 BusinessEntityID from Sales .
SalesPersonQuotaHistory where
SalesQuota = @0 )" StatementId="1" StatementCompId="1" StatementType="SELECT"
StatementSubTreeCost="0.00756866" StatementEstRows="1" StatementOptmLevel="FULL"
StatementOptmEarlyAbortReason="GoodEnoughPlanFound">
```

提示 除去使用sys.dm_exec_query_plan，你也可以使用sys.dm_exec_text_query_plan以文本格式返回查询计划。这个DMV没有sys.dm_exec_query_plan的输出大小限制，而且还可以减少你要运算的特定语句。

22

这个参数化的示例也可能会在SIMPLE参数化设置的情况下发生，只是FORCED设置增加了引发参数化的机会。

为了将数据库选项设置回SIMPLE，再一次使用ALTER DATABASE命令：

```
ALTER DATABASE AdventureWorks
SET PARAMETERIZATION SIMPLE
GO
```

解析

这个技巧演示了如何让数据库使用强制参数化，然后再设置回简单参数化。首先在数据库AdevntureWorks中使用系统目录视图sys.databases检查参数化状态。在这之后，使用ALTER DATABASE和SET PARAMETERIZATION FORCED命令。再一次检查系统目录视图sys.databases确认修改了这个选项。在这之后，使用DBCC FREEPROCCACHE命令清空过程缓存。

警告 由于DBCC FREEPROCCACHE删除了过程缓存中所有的计划，而这会对性能产生负面影响，因此只能在测试、非运行环境下的SQL Server实例中使用它。

下一步，在表HumanResources.Employee上执行查询，然后使用Sales.SalesPersonQuotaHistory表的子查询。然后对sys.dm_exec_cached_plans系统目录视图和sys.dm_exec_query_plan动态管理函数执行查询。结果显示了XML格式的SQL计划（结果可能会不同），在WHERE子句中使用了参数占位符： 646

```
where SalesQuota = @0
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

提示　关于SQL执行计划的更多内容请参阅第28章。

22.2.9　为事务启用读取一致性

SQL Server提供了两个新数据库选项来允许语句级别和事务级别的读取一致性：ALLOW_SNAPSHOT_ISOLATION和READ_COMMITTED_SNAPSHOT（在这个技巧之后将演示它们）。

注解　两个数据库选项都会在同一技巧中讨论，因为它们都会影响到读取一致性。但是不可以同时使用它们。它们是独立的选项。如果你希望事务级别的读取一致性，就使用ALLOW_SNAPSHOT_ISOLATION；如果要在语句级别实现读取一致性，就使用READ_COMMITTED_SNAPSHOT。

数据库选项ALLOW_SNAPSHOT_ISOLATION在事务级别启用了数据快照。当启用了ALLOW_SNAPSHOT_ISOLATION时，可以使用快照事务隔离级别读取数据的事务一致版本，将数据以事务开始时的数据来看待。使用了这个选项，数据读取不会阻塞数据修改。如果在读取快照数据时数据被修改了，并且尝试在快照事务中修改数据，那么这个修改尝试将不会被允许，并且SQL Server更新冲突检测支持将发出报警然后停止这个尝试。一旦启用了这个数据库设置，当在事务开始前使用SET TRANSACTION ISOLATION LEVEL指定了SNAPSHOT隔离快照时，就会启动快照隔离。

注解　ALLOW_SNAPSHOT_ISOLATION操作的示例，请看3.3节下的技巧讲解。

READ_COMMITTED_SNAPSHOT设置在独立语句级别启用了行版本控制。不管行是否修改，行版本控制保留tempdb中行的原始副本，存储当前数据库中行的最新版本。对于存在大量事务活动的数据库，希望确认tempdb是否有足够的空间来容纳行版本。READ_COMMITTED_SNAPSHOT设置在查询会话中的独立语句级别启用了行版本控制。当启用了READ_COMMITTED_SNAPSHOT时，不会在数据上加锁。使用行版本返回语句的数据，数据当作开始执行语句时数据的状态来看待。在语句执行时被读取的数据仍然允许其他操作对其更新，而且不像是快照隔离，不会有强制更新冲突检测报警，警告你数据在读取时被修改。一旦启用这个数据库选项，在语句执行之前，当使用默认读取提交的隔离级别或使用SET TRANSACTION ISOLATION LEVEL和READ COMMITTED执行查询时，就启动行版本控制。

使用这些选项的主要益处是在读取操作中减少锁定。如果应用程序需要实时的数据值，那么这两个选项就不是最佳选择了。但是如果数据的快照对于应用程序来说是可以接受的，则设置这些选项可
647 能是合适的。

启用这些选项的语法如下：

```
ALTER DATABASE database_name
   SET ALLOW_SNAPSHOT_ISOLATION {ON | OFF }
    | READ_COMMITTED_SNAPSHOT {ON | OFF }
```

这个命令有两个参数：数据库的名称以及启用或禁用的快照选项。这个技巧将演示为AdventureWorks数据库启用两个行版本控制选项。首先，查询sys.databases确认当前数据库设置：

```
SELECT snapshot_isolation_state_desc,
      is_read_committed_snapshot_on
FROM sys.databases
WHERE name = 'AdventureWorks'
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
snapshot_isolation_state_desc          is_read_committed_snapshot_on
OFF                                    0
```

下一步，使用ALTER DATABASE启用两个选项（但是没有必要同时启用两个选项，因为你可以选择启用读一致选项的一个类型，而不启用另一个）：

```
ALTER DATABASE AdventureWorks
SET ALLOW_SNAPSHOT_ISOLATION ON

ALTER DATABASE AdventureWorks
SET READ_COMMITTED_SNAPSHOT ON
```

下一步，再一次检验数据库设置，确认修改：

```
SELECT snapshot_isolation_state_desc,
      is_read_committed_snapshot_on
FROM sys.databases
WHERE name = 'AdventureWorks'
```

这个查询返回：

```
snapshot_isolation_state_desc          is_read_committed_snapshot_on
ON                                     1
```

执行如下查询，再次关闭这些选项：

```
ALTER DATABASE AdventureWorks
SET ALLOW_SNAPSHOT_ISOLATION OFF

ALTER DATABASE AdventureWorks
SET READ_COMMITTED_SNAPSHOT OFF
```

解析

这个技巧首先通过查询sys.databases检查AdventureWorks数据库的行版本控制的当前状态。在这之后，执行了两个ALTER DATABAES命令在数据库中启用快照隔离和读取提交的隔离级别。再一次查询系统目录视图sys.databases确认修改。要牢记，不要同时启用这两个选项——你可以决定是否同时希望语句和事务级别的读取一致性，同时，或是都不。

648

22.2.10 配置数据库恢复模式

完全数据库备份是数据库的完整的副本。另一方面，事务日志备份只从最近的完全备份或最近的事务日志备份开始备份事务日志。当备份完成时，SQL Server截断事务日志中不活动的部分。除了允许从事务日志备份的那点进行还原，事务日志备份也允许时点恢复和事务标志恢复。时点恢复允许将数据还原到一个指定的时间段，例如，将数据库还原到数据库修改操作之前或失败之前。事务标记恢复恢复到第一个实例的“标记的”事务并且包括这个事务内所做的更新。

注解 更多关于事务日志备份的信息请看第29章。

SQL Server提供了3个不同的恢复模式，它们定义是否可以使用事务日志备份，以及何种数据库活动将写入事务日志中。这3种恢复模式为FULL、BULK_LOGGED和SIMPLE。

- 当使用SIMPLE恢复模式时，在数据库备份后事务日志会自动被截断，删除了执行事务日志备份的能力。在这个恢复模式下，数据丢失的风险依赖完全或差异备份的时间安排——并且将不

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

可以执行事务日志备份提供的时点恢复。

- BULK_LOGGED恢复模式允许执行完全、差异以及事务日志备份，然而对于大容量操作只有最少的日志记录到事务日志中。这种恢复模式的好处在于大容量操作时减少了日志使用空间，然而作为代价，事务日志备份仅可以用于从事务日志备份末端进行恢复（时点恢复或标记事务是不允许的）。
- FULL恢复模式完整地记录所有事务活动，包括大容量操作。在这个最安全的模式下，所有还原选项都是可用的，包括时点事务日志还原、差异备份以及完全数据库备份。

修改数据库恢复模式的语法如下：

```
ALTER DATABASE database_name
SET RECOVERY { FULL | BULK_LOGGED | SIMPLE }
```

在这个技巧中，AdventureWorks数据库将设置为FULL恢复模式：

```
SELECT recovery_model_desc
FROM sys.databases
WHERE name = 'AdventureWorks'
GO

ALTER DATABASE AdventureWorks
SET RECOVERY FULL
GO

SELECT recovery_model_desc
FROM sys.databases
WHERE name = 'AdventureWorks'
```

649 这个查询返回：

```
recovery_model_desc
SIMPLE


recovery_model_desc
FULL
```

解析

当基于数据库model的恢复模式创建了数据库后，就启动了恢复模式。在创建了数据库之后，你可以随时使用ALTER DATABASE和SET RECOVERY修改恢复模式。

在这个技巧中，使用系统目录视图sys.databases检查AdventureWorks数据库的恢复模式。在确定了当前的数据库使用了SIMPLE模式之后，使用ALTER DATABASE和SET RECOVERY将数据库修改为FULL模式。

提示　*修改了数据库的恢复模式之后，对数据库执行完全备份是一个很好的习惯。*

22.2.11　配置页验证

当数据页写入物理磁盘时，如果被电源故障或其他物理问题打断就会发生磁盘错误。SQL Server有3种模式来控制和检测由磁盘错误引起的不完整的I/O事务，它包括：CHECKSUM、TORN_PAGE_DETECTION和NONE。

- CHECKSUM选项（数据库model中的默认值）基于整个数据页的内容将校验和值写入数据页头。如果页面被破坏或被部分地写入，SQL Server将会从页头与实际的页面内容中检测出异同。这

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

个选项提供最强的保护。

- TORN_PAGE_DETECTION选项（SQL Server早期版本中使用的主选项）通过每个数据页的512字节扇区的反转位检测数据页问题。当SQL Server读取的位值处于不正确状态时，就会识别出“残缺”的页面。
- 当选择NONE时，在分配新数据页时，不使用CHECKSUM或TORN_PAGE_DETECTION来控制，SQL Server也不会进行读取识别。

除非有很好的理由不去这么做（比如基准测试中无限制的查询性能的需求），否则最好还是保持默认选项CHECKSUM。尽管CHECKSUM比TORN_PAGE_DETECTION开销更大，但它有更全面的识别出数据页错误的能力。设置页验证模式的语法如下：

```
ALTER DATABASE database_name
SET PAGE_VERIFY { CHECKSUM | TORN_PAGE_DETECTION | NONE }
```

在这个技巧中，将AdventureWorks数据库修改为不进行页面验证：

```
SELECT page_verify_option_desc
FROM sys.databases
WHERE name = 'AdventureWorks'
GO

ALTER DATABASE AdventureWorks
SET PAGE_VERIFY NONE
GO

SELECT page_verify_option_desc
FROM sys.databases
WHERE name = 'AdventureWorks'
GO
```

这个查询返回：

```
page_verify_option_desc
CHECKSUM

page_verify_option_desc
NONE
```

现在将它置回到以前的状态：

```
ALTER DATABASE AdventureWorks
SET PAGE_VERIFY CHECKSUM
GO

SELECT page_verify_option_desc
FROM sys.databases
WHERE name = 'AdventureWorks'
GO
```

这个查询返回：

```
page_verify_option_desc

CHECKSUM
```

650

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

这个技巧首先通过查询AdventureWorks数据库中的系统目录视图sys.databases来确认当前页验证状态。在这之后，执行ALTER DATABASE和SET PAGE_VERIFY命令禁用页验证。然后再一次查询系统目录视图sys.databases确认所做的修改。

22.3　控制数据库访问和拥有关系

在下面两个技巧中，将讲到如何控制用户数据库的访问和拥有关系。首先，展示如何使用3种不同的状态来修改数据库的可访问性，它们是：联机、离线或紧急状态。随后的技巧演示了如何使用系统存储过程sp_changedbowner修改数据库的拥有者。

651

22.3.1　修改数据库状态为联机、离线或紧急状态

数据库有3种状态：联机、离线或紧急状态。

默认为联机状态（online state），意思是数据库是打开的并且是可用的。当为离线状态（offline state）时，数据是关闭的，并且不可以修改或被任何用户查询。在需要将数据文件移到新的物理位置，然后使用ALTER DATABASE修改数据文件新位置的元数据（将会在本章稍后演示）的情况下，你可能会希望将数据库置为离线。不像分离数据库，数据仍然会保持在SQL Server实例的元数据中，并且之后可以将其置为联机。

最后，如果数据库被破坏了，将数据库设置到紧急状态允许服务器角色登录名sysadmin到数据库的只读访问，允许查询所有仍可以访问的数据库对象（基于问题的种类）。

设置数据状态的语法如下：

```
ALTER DATABASE database_name
SET { ONLINE | OFFLINE | EMERGENCY }
```

这个技巧演示了如何将数据库置为离线，尝试读取，然后再将它置为联机。要牢记，如果在AdventureWorks数据库中存在活动的连接，除非强制踢掉它们（使用22.1.5节的技巧中讨论的技术来完成），否则你的命令不得不一直等待它们断开连接：

```
USE master
GO

ALTER DATABASE AdventureWorks
SET OFFLINE
GO

-- 尝试从表中读取
SELECT COUNT(*)
FROM AdventureWorks.HumanResources.Department
GO
```

这个查询返回：

```
消息942，级别14，状态4，第3行
由于数据库'AdventureWorks'离线，无法打开该数据库。
```

现在再一次将数据库置回到联机状态：

```
ALTER DATABASE AdventureWorks
SET ONLINE
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个技巧中，使用ALTER DATABASE和SET OFFLINE命令将AdventureWorks数据库置为离线。在将数据库置为离线之后，尝试查询数据库中的表，它会引发错误。然后使用ALTER DATABASE和SET ONLINE选项将数据库置为联机。

652

22.3.2 修改数据库拥有者

在这个技巧中，演示了如何使用系统存储过程sp_changedbowner修改既有数据库的拥有者。

这个系统存储过程的语法如下：

```
sp_changedbowner [ @loginame = ] 'login'
        [ , [ @map= ] remap_alias_flag ]
```

表22-10简要地描述了这个存储过程的参数。

表22-10 sp_changedbowner的参数

参 数	描 述
'login'	这个参数指定拥有数据库的SQL Server登录名。这个登录名不能已经映射到既有数据库用户（没有先删除这个用户）
remap_alias_flag	可选标志在早期SQL Server版本使用的引用别名的功能，允许将用户映射到数据库。将来的SQL Server版本上将会删除别名功能，因此不要使用它

22

这个技巧创建了新的登录名，然后将新登录名设为BookWarehouse数据库的拥有者：

```
CREATE LOGIN NewBossInTown WITH PASSWORD = 'HereGoesTheNeighborhood10'
GO

USE BookData
GO

EXEC sp_changedbowner 'NewBossInTown'
GO

SELECT p.name
FROM sys.databases d
INNER JOIN sys.server_principals p ON
   d.owner_sid = p.sid
WHERE d.name = 'BookData'
```

这个查询返回：

```
name
NewBossInTown
```

解析

拥有者从既有SQL Server登录名映射到数据库中的dbo用户上。一旦发生了这个操作，新的拥有者就拥有了执行所有特定于数据库的操作（例如，创建表、授予对象权限、删除数据等）的权限。

在这个技巧中，创建了新的登录名，并将数据库上下文切换到了BookWarehouse数据库。使用了系统存储过程sp_changedbowner设置新的登录名为拥有者：

```
EXEC sp_changedbowner 'NewBossInTow'
```

653

然后新的拥有者映射到dbo数据库用户上。设置好后，查询系统目录视图sys.databases和sys.

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

server_principals来确认拥有者已修改。

22.4　管理数据库文件和文件组

在下面一组技巧中将讲到如何管理数据库文件和文件组。具体将分别讲解以下操作。

- 添加数据和日志文件到既有的数据库。
- 从数据库中删除数据或日志文件。
- 在操作系统中重新分配数据或事务日志文件。
- 修改文件的逻辑名。
- 增加数据库文件的大小以及修改增长选项。
- 为既有数据库添加文件组。
- 为数据库设置默认文件组。
- 从数据库删除文件组。
- 将数据库或文件组设置为只读。

在后面的技巧中会演示如何使用ALTER DATABASE将数据或日志文件添加到既有的数据库。

22.4.1　为既有的数据库添加数据文件或日志文件

一旦创建了数据库，假定有足够的磁盘空间，就可以根据需要添加其他数据或事务日志。如果当前物理磁盘/阵列快满，或是希望通过在多个磁盘上分离I/O来提升性能，你可以扩展新的磁盘。这通常只是在计划将这些文件放置到单独的磁盘/阵列时将其他数据和日志文件添加到数据库时才有意义。将多个文件放置到相同的磁盘/阵列是不会提升性能的，并且可能只在计划为超大型数据库执行分离的文件或文件组备份时才有好处。

添加文件不需要将数据库设置为离线。使用ALTER DATABASE添加数据或事务日志文件的语法如下：

```
ALTER DATABASE database_name
{ADD FILE <filespec> [ ,...n ]
       [ TO FILEGROUP { filegroup_name | DEFAULT } ]
  | ADD LOG FILE <filespec> [ ,...n ] }
```

表22-11描述了这个语法的参数。

表22-11　ALTER DATABASE...ADD FILE的参数

参　数	描　述
database_name	定义既有数据库的名称
<filespec> [,...n]	指定一个或多个显式定义的数据文件添加到数据库
filegroup_name \| DEFAULT	指定文件组的逻辑名称。如果跟随关键字DEFAULT，这个文件组将为数据库的的默认文件组（意思是所有对象都将默认创建在这里）
[LOG ON { <filespec> [,...n] }]	指定一个或多个显式定义的事务日志文件添加到数据库

654

在这个技巧中，将新的数据和事务日志文件添加到数据库BookData中：

```
ALTER DATABASE BookData
ADD FILE
(  NAME = 'BookData2',
  FILENAME = 'C:\Apress\BD2.NDF' ,
  SIZE = 1MB ,
  MAXSIZE = 10MB,
  FILEGROWTH = 1MB )
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
TO FILEGROUP [PRIMARY]
GO

ALTER DATABASE BookData
ADD LOG FILE
(  NAME = 'BookData2Log',
  FILENAME = 'C:\Apress\BD2.LDF' ,
  SIZE = 1MB ,
  MAXSIZE = 5MB,
  FILEGROWTH = 1MB )
GO
```

解析

在这个技巧中，将新的数据和事务日志文件添加到数据库BookData中。ALTER DATABASE中使用ADD FILE命令添加数据文件，随后是文件的规格说明：

```
ALTER DATABASE BookData
ADD FILE
...
```

使用TO FILEGROUP子句指定添加新文件的文件组，随后在中括号中指定文件组名称：

```
TO FILEGROUP [PRIMARY]
GO
```

在技巧中的第二个查询，使用ALTER DATABASE和ADD LOG FILE命令添加新的事务日志文件：

```
ALTER DATABASE BookData
ADD LOG FILE
...
```

两种文件的添加都不需要将数据库置为离线。

22.4.2 从数据库删除数据或日志文件

这个技巧演示了如何从既有的数据库中删除数据或事务日志文件。如果需要通过在一个磁盘上创建新文件然后删除旧文件，将文件从一个磁盘 / 阵列重新分配到不同的磁盘 / 阵列，就可能希望进行这个操作。

删除文件（数据或事务日志）的语法如下：

```
ALTER DATABASE database_name
REMOVE FILE logical_file_name
```

表22-12描述了这个语法的参数。

表22-12 ALTER DATABASE...REMOVE FILE的参数

参　数	描　述
database_name	既有数据库的名称
logical_file_name	从数据库中删除文件的逻辑文件名称

这个技巧首先检查数据库BookData的逻辑文件名称，然后清空这个文件中的内容（这将数据移动到保留的数据文件中），最后从数据库中删除这个文件：

```
USE BookData
GO

SELECT name
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FROM sys.database_files

DBCC SHRINKFILE(BookData2, EMPTYFILE)

ALTER DATABASE BookData
REMOVE FILE BookData2
```

这个查询返回：

```
name
BookData
BookData_log
BookData2
BookData2Log

DbId    FileId    CurrentSize    MinimumSize    UsedPages    EstimatedPages
11      3         128            128            0            0

DBCC 执行完毕。如果 DBCC 输出了错误信息，请与系统管理员联系。
文件'BookData2'已删除。
```

解析

技巧首先切换到数据库BookData，基于对sys.database_files的查询从当前连接所在的数据库中返回所有逻辑文件名称。

656

不可以从数据库中删除主数据或主事务日志文件。也不可以删除包含数据或有活动事务的文件。

使用DBCC SHRINKFILE从要删除的文件中移走既有的数据。通过使用EMPTYFILE参数来完成它（本章稍后会介绍这个命令）。

在这之后，使用带有REMOVE FILE命令的ALTER DATABASE从数据库中删除文件。从数据库中删除文件也将从文件系统删除底层文件。

22.4.3　重新分配数据或事务日志文件

有时你可能发现为既有的数据库重新分配数据库文件是必要的。这样做的原因可能有多种——因为物理磁盘空间不足，或是要提升性能（将文件放置到分离的RAID阵列中）时可能需要这样做。

这个技巧演示了如何使用ALTER DATABASE命令修改数据库文件的位置。修改文件位置的语法如下：

```
ALTER DATABASE database_name
MODIFY FILE
{NAME = logical_file_name , FILENAME = 'new_physical_file_name_and_path')
```

表22-13描述了这个语法的参数。

表22-13　ALTER DATABASE...MODIFY FILE的参数

参　数	描　述
database_name	既有数据库的名称
logical_file_name	要重新分配的物理文件的逻辑文件名称
new_physical_file_name_and_path	新文件的路径和位置

在这个技巧中，将使用默认设置创建名为BookTransferHouse的新数据库。在这之后，将数据库置为离线，然后复制到服务器中新的位置上。完成移动后，使用ALTER DATABASE命令重新分配文件：

```
USE master
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
GO

-- 为这个示例创建默认的数据库
CREATE DATABASE BookTransferHouse
GO

ALTER DATABASE BookTransferHouse
SET OFFLINE
GO
```

现在将文件C:\Program Files\Microsoft SQL Server\MSSQL10.AUGUSTUS\MSSQL\DATA\BookTransferHouse.mdf移到C:\Apress目录。在这之后，执行如下操作：

```
ALTER DATABASE BookTransferHouse
MODIFY FILE
(NAME = 'BookTransferHouse', FILENAME = 'C:\Apress\BookTransferHouse.mdf')
GO
```

这个查询返回：

```
文件'BookTransferHouse'在系统目录中已修改。新路径将在数据库下次启动时使用。
```

657

最后，数据库必须置回到联机状态：

```
ALTER DATABASE BookTransferHouse
SET ONLINE
GO
```

解析

技巧首先使用默认选项创建新数据库。然后将数据库置为离线，在离线之后，将新数据文件复制到新的位置。

在移动完文件之后，通过使用ALTER DATABASE和MODIFY FILE语句通知SQL Server所做的修改。在这之后，通过使用ALTER DATABASE和SET ONLINE将数据库置回到联机状态。

22.4.4 修改文件的逻辑名称

修改数据库文件的逻辑名称时，可以不将数据库置为离线。数据库的逻辑名称不会影响数据库本身的功能，允许出于一致性和命名约定的原因而修改名称。例如，如果使用新的数据库名称从备份还原数据库，可能会希望逻辑名称匹配新的数据库名称。

修改逻辑文件名称的语法如下：

```
ALTER DATABASE database_name
{NAME = logical_file_name
    [ , NEWNAME = new_logical_name ] }
```

表22-14简要地描述了这个语法的参数。

表22-14 ALTER DATABASE...NEWNAME的参数

参 数	描 述
database_name	既有数据库的名称
logical_file_name	要重命名的逻辑文件名称
new_logical_name	新逻辑文件名称

这个技巧修改了BookTransferHouse数据库中数据文件BookTransferHouse的逻辑数据文件名称：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ALTER DATABASE BookTransferHouse
MODIFY FILE
(NAME = 'BookTransferHouse', NEWNAME = 'BookTransferHouse_DataFile1')
GO
```

这个查询返回：

```
文件名称'BookTransferHouse_DataFile1'已设置。
```

解析

这个技巧通过使用ALTER DATABASE和MODIFY FILE命令将逻辑文件名称BookTransferHouse修改为
658 BookTransferHouse_DataFile1。为了实现这个修改，命令使用了原始的NAME值和NEWNAME值。

22.4.5　增加数据库文件的大小和修改它的增长选项

前面的技巧演示了如何使用ALTER DATABASE和MODIFY FILE修改逻辑文件名称。通过MODIFY FILE命令也可以修改文件大小的设置。

语法如下：

```
ALTER DATABASE database_name
MODIFY FILE
(
NAME = logical_file_name
[ , SIZE = size [ KB | MB | GB | TB ] ]
[ , MAXSIZE = { max_size [ KB | MB | GB | TB ] |
UNLIMITED } ]
[ , FILEGROWTH = growth_increment [ KB | MB | % ] ]
)
```

表22-15简要地描述了这个语法的参数。

表22-15　ALTER DATABASE...MODIFY FILE的参数

参　数	描　述
database_name	既有数据库的名称
logical_file_name	要修改大小或增长选项的逻辑文件名称
size [KB \| MB \| GB \| TB]	基于选择的大小单位（千字节、兆字节、千兆字节、兆兆字节）的文件新的大小（必须比既有的大小要大）
{ max_size [KB \| MB \| GB \| TB] \| UNLIMITED }]	选择的大小单位的文件新的最大允许值，如果选择了UNLIMITED，文件可以一直增长直到磁盘空间已满
growth_increment [KB \| MB \| %]]	当需要空间时文件大小要增长的新的数量。可以指定千字节或兆字节数，或者是既有文件大小要增长的百分比。如果选择了0，文件将不会增长

在这个技巧中，文件增加到6MB的大小并且指定了文件最大允许值为10MB：

```
ALTER DATABASE BookTransferHouse
MODIFY FILE
(NAME='BookTransferHouse_DataFile1', SIZE=6MB, MAXSIZE=10MB)
```

解析

这个技巧使用ALTER DATABASE和MODIFY FILE修改指定文件的既有大小以及最大允许值。引用的NAME选项指定了要修改哪一个文件。其他两个选项，分别是SIZE和MAXSIZE，用来配置新文件大小和文件最
659 大允许值。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

22.4.6 为既有的数据库添加文件组

这个技巧演示了如何使用ALTER DATABASE为既有数据库添加文件组。创建文件组之后，可以为它添加一个或多个文件。

```
ALTER DATABASE database_name
ADD FILEGROUP filegroup_name
```

表22-16描述了这个语法的参数。

表22-16 ALTER DATABASE...ADD FILEGROUP的参数

参 数	描 述
database_name	既有数据库的名称
filegroup_name	新文件组的名称

这个技巧添加一个新文件组到BookTransferHouse数据库：

```
ALTER DATABASE BookTransferHouse
ADD FILEGROUP FG2
GO

ALTER DATABASE BookTransferHouse
ADD FILE
(  NAME = 'BW2',
  FILENAME = 'C:\Apress\BW2.NDF' ,
  SIZE = 1MB ,
  MAXSIZE = 50MB,
  FILEGROWTH = 5MB )
TO FILEGROUP [FG2]
GO
```

解析

这个技巧使用了ALTER DATABASE或ADD FILEGROUP将名为FG2的新文件组添加到既有数据库。然后使用ALTER DATABASE、ADD FILE和TO FILEGROUP命令将新文件添加到文件组。

22.4.7 设置默认文件组

这个技巧演示了如何将一个文件组改为默认文件组。意思是默认创建的所有新数据库对象要包含在文件组中（除非数据库对象在创建时显式地放置到不同的文件组）。

设置文件组为数据库的默认文件组的语法如下：

```
ALTER DATABASE database_name
MODIFY FILEGROUP filegroup_name
DEFAULT
```

这个技巧将BookTransferHouse数据库中的文件组FG2设置为默认文件组：

```
ALTER DATABASE BookTransferHouse
MODIFY FILEGROUP FG2 DEFAULT
```

这个查询返回：

```
文件组属性'DEFAULT'已设置。
```

22

660

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

这个技巧使用了ALTER DATABASE和MODIFY FILEGROUP将既有文件组修改为默认文件组。在新默认文件组的名称之后使用关键字DEFAULT。

22.4.8　删除文件组

这个技巧演示了如何删除一个用户定义的文件组。可以使用如下的语法删除空的文件组：

```
ALTER DATABASE database_name
REMOVE FILEGROUP filegroup_name
```

表22-17简要地描述了这个语法的参数。

表22-17　ALTER DATABASE...REMOVE FILEGROUP的参数

参　数	描　述
database_name	要删除的用户定义文件组所在的数据库的名称
filegroup_name	要删除的用户定义文件组的名称

在这个技巧中，将添加名为FG3的新文件组到BookTransferHouse数据库。然后在这个文件组中创建一个新的文件。在这之后，删除文件，然后删除用户定义文件组：

```
ALTER DATABASE BookTransferHouse
ADD FILEGROUP FG3
GO

ALTER DATABASE BookTransferHouse
ADD FILE
(  NAME = 'BW3',
  FILENAME = 'C:\Apress\BW3.NDF' ,
  SIZE = 1MB ,
  MAXSIZE = 10MB,
  FILEGROWTH = 5MB )
TO FILEGROUP [FG3]
GO

-- 现在，删除文件组中的文件
ALTER DATABASE BookTransferHouse
REMOVE FILE BW3
GO

-- 然后删除文件组
ALTER DATABASE BookTransferHouse
REMOVE FILEGROUP FG3
GO
```

661

这个查询返回：

```
文件'BW3'已删除。
文件组'FG3'已删除。
```

解析

当用户定义文件组为空时是可以删除的。在这个技巧中，首先使用ALTER DATABASE和REMOVE FILE命令清空用户定义文件组FG3的文件。清空了文件之后，使用ALTER DATABASE和REMOVE FILEGROUP从数据库中删除文件组。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

22.4.9 使数据库或文件组为只读

可以使用ALTER DATABASE将数据库或指定的用户定义文件组设为只读访问。将数据库或文件组设置为只读可以防止数据被修改，并且它经常用在静态报表数据库中。使用只读选项可以提升查询性能，由于事实上不允许修改数据库中或用户定义的文件组中的数据和对象（尽管这不能取代为数据和对象修改设置合适的安全权限），因此SQL Server不再需要锁定数据库中被查询的对象，因为这个原因，使用只读选项可以提升查询性能。

修改数据库可更新性的语法如下：

```
ALTER DATABASE database_name
SET { READ_ONLY | READ_WRITE }
```

这个语句的参数仅需要数据库名称，以及要设置的可更新性选项。

修改文件组可更新性的语法如下：

```
ALTER DATABASE database_name
MODIFY FILEGROUP filegroup_name
{ READ_ONLY | READ_WRITE }
```

这个语法块涉及数据库名称、文件组和可更新性选项。

这个技巧演示了将整个BookTransferHouse设置为只读模式，然后再将它设置回可读写模式（可以再次修改）。在这之后，技巧演示了设置指定文件组的可更新性：

```
USE master
GO

-- 将数据库设置为只读
ALTER DATABASE BookTransferHouse
SET READ_ONLY
GO

-- 再次允许更新
ALTER DATABASE BookTransferHouse
SET READ_WRITE
GO

-- 添加新文件组
ALTER DATABASE BookTransferHouse
ADD FILEGROUP FG4
GO

-- 将文件添加到文件组
ALTER DATABASE BookTransferHouse
ADD FILE
(  NAME = 'BW4',
  FILENAME = 'C:\Apress\BW4.NDF' ,
  SIZE = 1MB ,
  MAXSIZE = 50MB,
  FILEGROWTH = 5MB )
  TO FILEGROUP [FG4]
  GO

-- 将指定文件组设置为只读模式
ALTER DATABASE BookTransferHouse
MODIFY FILEGROUP FG4 READ_ONLY
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
-- 再次允许更新
ALTER DATABASE BookTransferHouse
MODIFY FILEGROUP FG4 READ_WRITE
GO
```

解析

这个技巧演示了修改数据库和指定文件组的可更新性。使用ALTER DATABASE和SET READ_ONLY修改数据库。使用SET READ_WRITE再次允许更新。技巧中最后两个查询使用ALTER DATABASE和MODIFY FILEGROUP更新指定文件组的可更新性。

22.5　查看和管理数据库空间使用

本章的最后一组技巧讲解如何管理和查看数据库磁盘空间的使用情况。你也将学习如何收缩整个数据库，或只是收缩其中的单个文件，这都根据需求而定。随后的技巧演示了如何使用系统存储过程sp_spaceused查看空间使用情况。

22.5.1　查看数据库空间使用情况

这个技巧演示了如何使用系统存储过程sp_spaceused显示数据库数据磁盘空间使用情况。为了查看事务日志的使用情况，也将演示DBCC SQLPERF命令。

sp_spaceused的语法如下：

```
sp_spaceused [[ @objname = ] 'objname' ]
[,[ @updateusage = ] 'updateusage' ]
```

表22-18简要地描述了这个存储过程的参数。

663

表22-18　sp_spaceused的参数

参　数	描　述
'objname'	这个参数定义了查看空间使用情况的可选对象名称（例如表）。如果没有指定，就返回整个数据库的空间使用情况的信息
'updateusage'	在指定的对象中使用这个参数，可使用的值为true或false。如果为true，则使用DBCC UPDATEUSAGE命令更新系统表中空间使用情况的信息

DBCC SQLPERF的语法如下：

```
DBCC SQLPERF ( LOGSPACE )
[WITH NO_INFOMSGS ]
```

表22-19简要地描述了这个DBCC命令的参数。

表22-19　DBCC SQLPERF的参数

参　数	描　述
LOGSPACE	这是唯一允许的文本变量。并且当指定它时，返回SQL Server实例的事务日志空间信息
WITH NO_INFOMSGS	当包含在命令中时，WITH NO_INFOMSGS从DBCC的输出中取消信息性的消息

在这个技巧中，查看AdventureWorks数据库的数据库以及事务日志空间：

```
USE AdventureWorks
GO

EXEC sp_spaceused
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回：

```
database_name      database_size    unallocated space
AdventureWorks     181.94 MB        48.32 MB

reserved     data         index_size     unused
134776 KB    83872 KB     44912 KB       5992 KB
```

下一步，显示整个SQL Server实例的事务日志信息：

```
DBCC SQLPERF ( LOGSPACE )
```

这个查询返回如下内容（你的结果可能不同）：

Database Name	Log Size (MB)	Log Space Used (%)	Status
master	0.7421875	47.36842	0
tempdb	0.7421875	60.85526	0
model	0.4921875	81.74603	0
msdb	0.4921875	82.53968	0
AdventureWorks	1.992188	18.97059	0
BookStore	0.484375	44.75806	0
BookStoreArchive	0.4921875	46.9246	0
BookStoreInternational	0.4921875	49.80159	0
BookMart	0.484375	45.66532	0
TestAttach	0.484375	48.18548	0
BookData	1.476563	43.02249	0
BookStoreArchive_Ukrainian	0.9921875	53.98622	0
BookTransferHouse	0.484375	87.70161	0

664

22

解析

在这个技巧中，使用系统存储过程sp_spaceused和DBCC SQLPERF命令返回AdventureWorks数据库的空间使用情况。

在sp_spaceused结果中，database_size列显示了当前数据库的大小（包括数据和日志文件）。unallocated space列显示了数据库中未使用的空间，以及数据库对象使用的reserved列空间总量。data列显示了对象数据使用的空间总量，以及index_size索引使用的空间总量。

DBCC SQLPERF的输出返回了SQL Server实例中所有数据库的数据，以兆字节显示日志大小，以及日志文件的当前活动与非活动日志信息的使用空间的百分比信息。

22.5.2 收缩数据库或数据库文件

在这个技巧中，演示了如何使用DBCC SHRINKDATABASE收缩整个数据库，或者使用DBCC SHRINKFILE收缩指定的数据库文件。在这个技巧中，要牢记收缩数据库和数据文件是开销较大的操作，会带来碎片，只有当确定必要时才执行它。

当启用自动增长时，由于重建索引或数据修改活动，数据库文件会扩展。由于数据修改和索引重建，在数据库中可能需要额外的空间。如果你不需要释放未使用的空间，应该允许数据库保留它。但是，如果你的确需要未使用的空间并且希望释放它，就可以使用DBCC SHRINKDATABASE或DBCC SHRINKFILE命令。

使用DBCC SHRINKDATABASE命令收缩数据库中的数据和日志文件。

注解 这个命令将收缩单个数据文件（MDF、NDF），但是会将多个事务日志文件（LDF）收缩为一个连续的事务日志文件。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

语法如下：

```
DBCC SHRINKDATABASE
( 'database_name' | database_id | 0
     [ ,target_percent ]
     [ , { NOTRUNCATE | TRUNCATEONLY } ]
)
[ WITH NO_INFOMSGS ]
```

表22-20中描述了这个命令的参数。

665

表22-20　DBCC SHRINKDATABASE的参数

参　数	描　述
'database_name' \| database_id \| 0	可以指定一个特定的数据库名称或系统数据库ID去收缩数据库。如果指定为0，将收缩在查询会话连接到的当前数据库
target_percent	在收缩事件完成后指定的保留在数据库文件的自由空间的目标百分比
NOTRUNCATE \| TRUNCATEONLY	NOTRUNCATE为了创建自由空间执行所需要的数据移动，但是在文件中保留了自由空间而不会将它释放给操作系统。如果未指定NOTRUNCATE，释放的自由空间将还给操作系统。TRUNCATEONLY不通过在文件中重新分配数据而释放空间。如果未指定，将在文件中重新分配数据页去释放空间，这会引发较大的I/O需求
WITH NO_INFOMSGS	这个参数从DBCC命令的返回中取消信息性消息

使用DBCC SHRINKFILE命令收缩数据库中的指定数据库文件。语法如下：

```
DBCC SHRINKFILE
(
   { ' file_name ' | file_id }
   { [ , EMPTYFILE]
   | [ [ , target_size ] [ , { NOTRUNCATE | TRUNCATEONLY } ] ]
   }
)
[ WITH NO_INFOMSGS ]
```

表22-21描述了这个命令的参数。

表22-21　DBCC SHRINKFILE的参数

参　数	描　述
'file_name' \| file_id	这个选项定义了指定要收缩的逻辑文件名称或文件ID
EMPTYFILE	这个参数为了可以使用ALTER DATABASE和REMOVE FILE删除数据库文件，从文件中移走所有数据
target_size	这个选项指定留在数据库文件中的自由空间（以MB为单位）。保持空白空间，通知SQL Server将空间释放为默认文件大小
NOTRUNCATE \| TRUNCATEONLY	NOTRUNCATE重新分配文件中已分配的数据页到文件的前端，但是不会释放空间给操作系统。当使用NOTRUNCATE时将忽略目标文件大小，但是只有在文件末端发现自由空间时才会这样做。不会重新排列或重新分配数据页。使用TRUNCATEONLY也会忽略目标文件大小。如果你一定要通过最少影响数据库性能的方式（在活动的数据库中重新排列数据页会引起性能问题，比如说减慢查询响应时间）在数据库文件中释放空间，可以使用这个选项
WITH NO_INFOMSGS	这个参数从DBCC命令的返回中取消信息性消息

666

在这个技巧中，将通过使用ALTER DATABASE...MODIFY FILE分配额外的空间来扩展AdventureWorks

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的文件，然后使用两个DBCC文件和数据库收缩命令进行收缩操作。在第一个示例中，AdventureWorks的数据和事务日志文件都扩展为更大的大小，然后使用一个DBCC操作进行收缩操作：

```
ALTER DATABASE AdventureWorks
MODIFY FILE (NAME = AdventureWorks2008_Data , SIZE= 250MB)
GO

ALTER DATABASE AdventureWorks
MODIFY FILE (NAME = AdventureWorks2008_Log , SIZE= 500MB)
GO
```

然后为AdventureWorks数据库使用系统存储过程sp_spaceused返回空间使用情况：

```
USE AdventureWorks
GO

EXEC sp_spaceused
GO
```

这个查询返回：

```
database_name    database_size    unallocated space
AdventureWorks   750.00 MB        118.38 MB

reserved    data        index_size    unused
134776 KB   83872 KB    44912 KB      5992 KB
```

下一步，使用DBCC SHRINKDATABASE命令减小数据库的大小：

```
DBCC SHRINKDATABASE ('AdventureWorks', 10)
```

这个查询返回如下内容（结果可能不同）：

```
DbId    FileId    CurrentSize    MinimumSize    UsedPages    EstimatedPages
5       1         18664          15360          16752        16752
5       2         8224           256            8224         256
```

在这个技巧的第二个示例中，只扩展了事务日志文件，然后使用DBCC SHRINKFILE进行收缩操作：

```
ALTER DATABASE AdventureWorks
MODIFY FILE (NAME = AdventureWorks2008_Log , SIZE= 150MB)
GO
```

然后使用系统存储过程sp_spaceused返回AdventureWorks数据库的空间使用情况：

```
USE AdventureWorks
GO

EXEC sp_spaceused
GO
```

这个查询返回：

```
database_name    database_size    unallocated space
AdventureWorks   295.81 MB        14.52 MB

reserved      data        index_size    unused
134440 KB     83864 KB    44912 KB      5664 KB
```

下一步，使用DBCC SHRINKDATABASE命令减小文件的大小：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DBCC SHRINKFILE ('AdventureWorks2008_Log', 100)
```

这个查询返回：

DbId	FileId	CurrentSize	MinimumSize	UsedPages	EstimatedPages
5	2	13696	256	13696	256

解析

DBCC SHRINKDATABASE收缩数据库中的数据和日志文件。在这个技巧中，AdventureWorks中的数据和日志文件都增大到更大的大小。在这之后，使用DBCC SHRINKDATABASE命令减小数据库的大小到目标自由空间大小的10%：

```
DBCC SHRINKDATABASE ('AdventureWorks', 10)
```

在执行完之后，命令返回的结果集显示了当前大小（以8KB页为单位）、最小大小（以8KB页为单位）、当前使用的8KB页以及估计的SQL Server可以将文件收缩到的8KB页。

在技巧的第二部分，演示了DBCC SHRINKFILE。DBCC SHRINKFILE与DBCC SHRINKDATABASE非常相似，只是使用它可对单个的数据和日志文件大小进行收缩，而不是数据库中所有的文件。在这个技巧中，扩展了AdventureWorks事务日志文件，然后收缩到指定的大小（以兆字节为单位）：

```
DBCC SHRINKFILE ('AdventureWorks_Log', 100)
```

这个命令通过删除不活动的虚拟日志文件来收缩物理文件。虚拟日志文件（VLF）以最小256KB或更大的大小为范围，是事务日志截断操作的单位，并且以记录形式写入事务日志中。

在事务日志中的是日志中“活动的”逻辑部分。这是包括活动事务的事务日志中的区域。这个活动的部分通常不会与物理文件界限匹配。但是将会从VLF到VLF进行“循环”。一旦VLF不再包含活动的事务，它可以通过BACKUP LOG操作或自动的系统截断操作截断。这个截断操作不会减少事务日志文件的大小，它只是使VLF对于新日志记录是可用的。

DBCC SHRINKFILE或DBCC SHRINKDATABASE最好的成果就是从物理文件的末端删除不活动的VLF。SQL Server也将尝试向物理文件的前端添加“假”行插入到活动的逻辑日志中——所以有时在第一次执行DBCC SHRINKFILE命令之后发出BACKUP LOG，然后再次发出BACK SHRINKFILE命令，将允许释放原始请求的空间。

668

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第23章 数据库完整性和优化

在前面的章节中，展示了如何创建、配置、修改和删除数据库。本章将展示如何使用T-SQL语言来维护数据库，其中包括使用ALTER INDEX命令重建索引或碎片整理，以及用DBCC命令帮助标识数据库完整性问题。本章还会演示SQL Server 2008新引入的堆重建技术(没有聚集索引的表)。

在以前的SQL Server版本中，DBCC命令和DBCC CHECKDB相似，是资源密集型的，并且如果在繁忙的SQL Server实例中执行可能会对性能产生不良影响。微软增强了某些DBCC命令，使用目标数据的内部数据库快照，而不是使用表或数据库锁。某些命令也能比以前版本的SQL Server更彻底地完成检查工作。

警告 某些在本章中介绍的DBCC命令有REPAIR选项。微软建议从最近的完好备份中恢复数据库来解决数据库完整性问题，而不是借助REPAIR选项。如果从备份中恢复不能作为一个选择，那么REPAIR选项只能在最后不得已的时候使用；使用了REPAIR选项后，就会发生数据丢失，并且问题可能依然存在。

本章包括可以周期性运行来检查数据库完整性问题的一些技巧。运行周期性的检查（每日、每周等）可标识出数据库各个地方发生的内部错误。

当数据库内的数据被修改后，表和索引会产生碎片。聚集或非聚集索引产生的碎片越多，为了完成相同的查询请求，数据库引擎需要返回的页面就越多。本章的最后两个技巧将指出如何周期性地使用T-SQL重建这些索引或对其进行碎片整理。

23.1 数据库检查

数据库完整性错误是非常少见的，但还是有可能会发生。下面两个技巧将介绍用来验证和检查数据库内部问题的命令。你将学习到如何使用DBCC CHECKALLOC检查页面在数据库中的使用和分配情况。你也将学习到如何使用DBCC CHECKDB命令检查数据库对象的完整性。

23.1.1 使用 DBCC CHECKALLOC 检查磁盘空间分配结构的一致性

DBCC CHECKALLOC检查页面在数据库中的使用和分配情况，并且报告任何发现的错误（这个命令也会自动执行DBCC CHECKDB——因此如果周期性运行CHECKDB，就没有必要再运行CHECKALLOC了）。

语法如下：

```
DBCC CHECKALLOC
(
    [ 'database_name' | database_id | 0 ]
        [ , NOINDEX
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    |
    { REPAIR_ALLOW_DATA_LOSS
    | REPAIR_FAST
    | REPAIR_REBUILD
    } ]
)
    [ WITH { [ ALL_ERRORMSGS ]
             [ , NO_INFOMSGS ]
             [ , TABLOCK ]
             [ , ESTIMATEONLY ]
           }
    ]
```

表23-1描述了这个命令的参数。

表23-1　DBCC CHECKALLOC参数

参　数	描　述
'database_name' \| database_id \| 0	这个参数定义要检查错误的数据库名称或数据库ID。当指定为0时，使用当前数据库
NOINDEX	当指定该参数时，不会检查非聚集索引。这是对DBCC CHECKALLOC没有影响的向后兼容的选项
REPAIR_ALLOW_DATA_LOSS \| REPAIR_FAST \| REPAIR_REBUILD	请参看本章开始时关于使用repair选项的警告。REPAIR_ALLOW_DATA_LOSS会尝试修复表或索引视图，在处理过程中也会带来丢失数据的风险。REPAIR_FAST和REPAIR_REBUILD为向后兼容而保留
ALL_ERRORMSGS	当指定ALL_ERRORMSGS时，所有发现的错误都会显示出来。如果不指定这个选项，则最多显示200条错误消息
NO_INFOMSGS	NO_INFOMSGS取消DBCC输出中的所有信息性消息
TABLOCK	当指定它时，不使用内部数据快照而使用排他表锁，这样就会潜在地降低数据库的查询并发性
ESTIMATEONLY	这个参数给出执行这个命令的tempdb数据库估计需要的空间

670

在这个简短的技巧中，将检查AdventureWorks数据库中数据页使用和分配情况的错误：

```
DBCC CHECKALLOC ('AdventureWorks')
```

这个命令会返回如下的（部分）结果。它包括页面使用的情况以及所有索引的分区。信息的关键部分在最后一行，你会在这里看到发生的分配错误和一致性错误数量的报告：

```
AdventureWorks的DBCC结果。
*****************************************************************
表sys.sysrscols                    对象ID 3.
索引ID 1，分区ID 196608，分配单元ID 196608 (类型为In-row data)。FirstIAM (1:86)。Root (1:87)。Dpages 0。
索引ID 1，分区ID 196608，分配单元ID 196608 (类型为In-row data)。使用了2个专用区中的19页。
总区数为2。
*****************************************************************
表sys.sysrowsets                   对象ID 5.
索引ID 1，分区ID 327680，分配单元ID 327680 (类型为In-row data)。
FirstIAM (1:131)。Root (1:234)。Dpages 0.
索引ID 1，分区ID 327680，分配单元ID 327680 (类型为In-row data)。
使用0个专用区中的5页。总区数为0。
……
    对象ID 2105058535，索引ID 0，分区ID 72057594038779904，分配单元ID 72057594039697408(类型为In-row data)，
数据区数98，页数787，混合区页数9。
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    对象ID 2105058535，索引ID 2，分区ID 72057594044547072，分配单元ID 72057594046709760(类型为In-row data)，
索引区数0，页数5，混合区页数5。
    对象ID 2117582582，索引ID 1，分区ID 72057594045857792，分配单元ID 72057594048151552(类型为In-row data)，
索引区数0，页数2，混合区页数2。
在此数据库中，总区数= 2864，已用页数= 22190，保留页数= 22904。
        此数据库中(混合区数= 91，混合页数= 720)。
CHECKALLOC在数据库'AdventureWorks'中发现0个分配错误和0个一致性错误。
DBCC执行完毕。如果DBCC输出了错误信息，请与系统管理员联系。
```

解析

在这个简短的技巧中，使用DBCC CHECKALLOC命令检查AdventureWorks数据库中除了FILESTREAM数据之外的所有数据库页和内部结构的分配情况。返回信息数据，其中包括内部页面信息、分区数量以及页面。在命令最后，报告了所有分配和一致性错误（在这个例子中，没有发现错误）。

在执行DBCC CHECKALLOC时，为了在操作过程中保证事务一致性，创建了内部数据库快照。如果因为某些原因不能创建数据库快照，或指定了TABLOCK，那么在执行这个命令时将获取排他数据库锁（这将潜在地影响数据库查询并发性）。除非你有足够的理由不去这样做，否则应该允许SQL Server去生成内部数据库快照，这样就不会影响数据库的并发性了。 671

23.1.2 使用 DBCC CHECKDB 检查所有数据库的分配和结构完整性

DBCC CHECKDB命令检查数据库中对象的完整性。对数据库周期性地运行DBCC CHECKDB是一个很好的维护经验。每周执行一次通常就足够了，但是，最佳的频率完全依赖于数据库的活动和容量。如果可能，应该在低数据库活动性期间执行DBCC CHECKDB。这样做可以让DBCC CHECKDB更快地完成，并且避免因它带来的开销减慢其他进程的运行。

就像我在本章中描述过的其他命令，当这个命令执行时，在操作期间会创建内部数据库快照来维护事务一致性。如果因某些原因不能创建数据库快照（或指定了TABLOCK选项），则检查表时保持共享表锁，并且分配检查时会使用排他数据库锁。

作为它执行的一部分，DBCC CHECKDB执行在本章讨论到的其他DBCC命令，包括DBCC CHECKTABLE、DBCC CHECKALLOC和DBCC CHECKCATALOG。除了这些内容，CHECKDB还会验证Service Broker 数据索引视图和表以及文件系统目录的FILESTREAM链接一致性。

DBCC CHECKDB的语法如下：

```
DBCC CHECKDB
(
        'database_name' | database_id | 0
    [ , NOINDEX
    | { REPAIR_ALLOW_DATA_LOSS
    | REPAIR_FAST
    | REPAIR_REBUILD
    } ]
)
    [ WITH         {
            [ ALL_ERRORMSGS ]
            [ , [EXTENDED_LOGICAL_CHECKS] ]
            [ , [ NO_INFOMSGS ] ]
            [ , [ TABLOCK ] ]
            [ , [ ESTIMATEONLY ] ]
            [ , { PHYSICAL_ONLY | DATA_PURITY } ]
     }
]
```

这个命令的参数看起来非常像本章前面讲到的命令，表23-2对此进行了描述。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表23-2　DBCC CHECKDB参数

参　数	描　述
'database_name' \| database_id \| 0	这个参数定义要检查错误的数据库名称或数据库ID。当指定为0时，使用当前数据库
NOINDEX	当指定该参数时，不会检查非聚集索引
REPAIR_ALLOW_DATA_LOSS \| REPAIR_FAST \| REPAIR_REBUILD	请参看本章开始时关于使用repair选项的警告。REPAIR_ALLOW_DATA_LOSS会尝试修复表或索引视图，在处理过程中也会带来丢失数据的风险。REPAIR_FAST和REPAIR_REBUILD为向后兼容而保留，而且REPAIR_ REBUILD执行修复不会有丢失数据的风险
ALL_ERRORMSGS	当指定ALL_ERRORMSGS时，所有发现的错误都会显示出来。如果不指定这个选项，则最多显示200条错误消息
EXTENDED_LOGICAL_CHECKS	当选定EXTENDED_LOGICAL_CHECKS时，它启用对空间和XML索引视图的逻辑一致性检查。这个选项会显著地影响性能，要小心使用它
NO_INFOMSGS	NO_INFOMSGS取消所有DBCC输出的信息性消息
TABLOCK	当定它时，不使用内部数据快照而使用排他数据库锁，这样就会潜在地降低数据库的查询并发性
ESTIMATEONLY	这个参数给出执行这个命令tempdb数据库估计需要的空间
PHYSICAL_ONLY \| DATA_PURITY	PHYSICAL_ONLY参数限制只进行物理的完整性检查，忽略逻辑的检查。如果选定了DATA_PURITY，是为升级的数据库使用的（SQL Server 2005以前的版本），这会指示DBCC CHECKDB去检测没有遵循数据类型的列值（例如可能会有bigint大小的值存储在整型值中）。一旦清理了升级的数据库中不正常的值，SQL Server将继续维护列值完整性

672

先不管所有这些语法选项，这个命令普通的执行形式几乎是最简单的。这个简短的技巧在AdventureWorks数据库上执行DBCC CHECKDB。要彻底地对数据库的完整性和数据进行检验，默认值通常就适用：

```
DBCC CHECKDB('AdventureWorks')
```

这个命令将返回如下信息性结果，它详细地对数据库中的数据库对象进行了评估，包括行数量、页数量，以及最重要的——分配错误或一致性错误的数量：

```
AdventureWorks的DBCC 结果。
Service Broker消息9675，状态1：已分析的消息类型：14。
Service Broker消息9676，状态1：已分析的服务约定：6。
……
sys.sysrowsetcolumns的DBCC结果。
对象'sys.sysrowsetcolumns'的9页中有1301行。
sys.sysrowsets的DBCC结果。
……
对象'Person.AddressType'的1页中有6行。
Production.ProductSubcategory的DBCC结果。
对象'Production.ProductSubcategory'的1页中有37行。
AWBuildVersion的DBCC结果。
对象'AWBuildVersion'的1页中有1行。
Production.TransactionHistoryArchive的DBCC结果。
对象Production.TransactionHistoryArchive对象的620页中有89253行。
……
CHECKDB 在数据库'AdventureWorks' 中发现0个分配错误和0个一致性错误。
DBCC 执行完毕。如果DBCC输出了错误信息，请与系统管理员联系。
```

673

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个技巧中使用DBCC CHECKDB对AdventureWorks数据库进行了彻底的完整性检查，在括号内是数据库的名称：

```
DBCC CHECKDB('AdventureWorks')
```

这个命令返回一些信息行，包括最终的分配错误或一致性错误的数量信息。

警告 正像在本章开始时警告过的，你应该知道如果DBCC遇到错误，微软建议从最近的完好的备份中恢复数据库解决数据库完整性问题，而不是借助REPAIR选项。如果从备份中恢复不能作为一个选择，REPAIR选项只能用作最后不得已的办法。使用了REPAIR选项后，就会发生数据丢失，并且问题可能依然存在。

23.2 表和约束

后面的一组技巧演示了用来在约束和表级别验证完整性的DBCC命令。特别地，我将演示如何使用下面的命令。

- DBCC CHECKFILEGROUP，它和DBCC CHECKDB非常相似，但限制了只在指定的文件组上的对象进行完整性和分配检查。
- DBCC CHECKTABLE，它用来对指定的表或索引视图标识出所有完整性问题。
- DBCC CHECKCONSTRAINTS，它会对指定表或约束的所有CHECK或违反约束的情况报警。

最后，我将介绍如何使用DBCC CHECKCATALOG命令检查系统表中和系统表之间的一致性。

674

23.2.1 使用 DBCC CHECKFILEGROUP 检查文件组中所有表的分配和结构完整性

DBCC CHECKFILEGROUP命令和DBCC CHECKDB非常相似，但限制了只在指定的一个文件组上的对象进行完整性和分配检查。对于超大型数据库（VLDB），执行DBCC CHECKDB操作可能在时间上是不允许的。作为替代，如果在数据库中使用用户定义的文件组，可以使用DBCC CHECKFILEGROUP命令进行每周的（或周期性的）检查——在不同的天中对文件组展开检查。

当执行这个命令时，在操作期间会创建内部数据库快照去维护事务一致性。如果由于某种原因而不能创建数据库快照（或指定了TABLOCK选项），命令会在检查表时生成共享表锁，在分配检查时生成排他数据库锁。

再一次提醒，如果DBCC CHECKDB发现错误，微软建议通过从最近完好的数据库备份中恢复来解决发现的问题。不像本章中其他的DBCC命令，DBCC CHECKFILEGROUP没有修复选项（总之微软不再建议使用修复选项）。

语法如下：

```
DBCC CHECKFILEGROUP
(
[ { 'filegroup' | filegroup_id | 0 } ]
[ , NOINDEX ]
)
    [ WITH
        {
            [ ALL_ERRORMSGS | NO_INFOMSGS ]
            [ , [ TABLOCK ] ]
            [ , [ ESTIMATEONLY ] ]
        }
          ]
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表23-3描述了这个命令的参数。

表23-3　DBCC CHECKFILEGROUP参数

参　数	描　述
'filegroup' \| filegroup_id \| 0	这个参数定义要检查的文件组名称或文件组ID。当指定为0时，使用主文件组
NOINDEX	当指定该参数时，不会对非聚集索引进行完整性检查
ALL_ERRORMSGS	当选定ALL_ERRORMSGS时，输出中的所有错误都将显示，而没有默认的200条消息的限制
NO_INFOMSGS	NO_INFOMSGS取消所有DBCC输出的信息性消息
TABLOCK	当指定它时，使用排他数据库锁而不用内部数据库快照（使用这个选项降低了与其他数据库的查询并发性，但是会提高DBCC执行的速度）
ESTIMATEONLY	给出执行这个命令估计的tempdb数据库需要的空间

675

在这个技巧中，检查AdventureWorks数据库中主文件组的完整性：

```
USE AdventureWorks
GO
DBCC CHECKFILEGROUP('PRIMARY')
```

这个命令返回如下部分结果：

```
AdventureWorks的DBCC 结果。
sys.sysrowsetcolumns的DBCC 结果。
对象'sys.sysrowsetcolumns' 的9 页中有1301 行。
sys.sysrowsets的DBCC 结果。
对象'sys.sysrowsets' 的2 页中有248 行。
sysallocunits的DBCC 结果。
……
对象'Sales.SalesReason' 的1 页中有10 行。
Sales.Individual的DBCC 结果。
对象'Sales.Individual' 的3082 页中有18484 行。
Sales.SalesTaxRate的DBCC 结果。
……
CHECKFILEGROUP 在数据库'AdventureWorks' 中发现0 个分配错误和0 个一致性错误。
DBCC 执行完毕。如果DBCC 输出了错误信息，请与系统管理员联系。
```

解析

在这个技巧中，对AdventureWorks数据库中PRIMARY文件组中所有对象的分配和结构完整性进行了检查。结果输出显示了文件组对象的行和页面数量，以及发现的分配错误和一致性错误的总数（在末尾报告）。像其他DBCC命令一样，要了解存在的问题，倒数第二行到最后一行是最关键的：

```
CHECKFILEGROUP在数据库'AdventureWorks'中发现0个分配错误和0个一致性错误。
DBCC执行完毕。如果DBCC输出了错误信息，请与系统管理员联系。
```

23.2.2　使用 DBCC CHECKTABLE 检查表和索引视图的数据完整性

为了标识出指定表或索引视图的问题，可以使用DBCC CHECKTABLE命令。如果想对数据库中所有表和索引视图进行检查，可以使用DBCCCHECKDB，它在数据库中所有表上执行DBCC CHECKTABLE命令。

当执行DBCC CHECKTABLE时，在操作期间会创建一个内部数据库快照去维护事务一致性。如果因为某些原因不能创建数据库快照，则会在目标表或索引视图应用共享表锁（这样会潜在地降低目标对象

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

上的数据库查询并发性）。

语法如下：

```
DBCC CHECKTABLE
(
        table_name | view_name
    [ , { NOINDEX | index_id }
      |, { REPAIR_ALLOW_DATA_LOSS | REPAIR_FAST | REPAIR_REBUILD }
    ]
)
    [ WITH
        { ALL_ERRORMSGS ]
          [ , EXTENDED_LOGICAL_CHECKS ]
          [ , NO_INFOMSGS ]
          [ , TABLOCK ]
          [ , ESTIMATEONLY ]
          [ , { PHYSICAL_ONLY | DATA_PURITY } ]
        }
]
```

表23-4描述了这个命令的参数。

表23-4 DBCC CHECKTABLE参数

参 数	描 述
'table_name' \| 'view_name'	这个参数定义要检查的表或索引视图
NOINDEX	这个关键字指示命令不去检查非聚集索引
index_id	这个参数指定要检查的索引的ID（如果你要检查指定索引的话）
REPAIR_ALLOW_DATA_LOSS \| REPAIR_FAST \| REPAIR_REBUILD	请参看在本章开始时对使用REPAIR选项的警告。REPAIR_ALLOW_ DATA_LOSS尝试修复表或索引视图，在处理过程中存在丢失数据的风险。REPAIR_FAST已不再使用，保留它只是为了向后兼容。REPAIR_REBUILD会在不丢失数据的情况下进行索引修复及重建
ALL_ERRORMSGS	当指定ALL_ERRORMSGS时，在命令执行过程中所有发现的错误都会显示
EXTENDED_LOGICAL_CHECKS	当指定EXTENDED_LOGICAL_CHECKS，它启用对空间和XML索引及索引视图的逻辑一致性检查。这个选项会显著地影响性能，要小心使用它
NO_INFOMSGS	NO_INFOMSGS取消DBCC输出的所有信息性消息
TABLOCK	当指定它时，在表上应用共享表锁而不是内部数据快照。使用这个选项会降低在表或索引视图上执行其他数据库查询的并发性
ESTIMATEONLY	给出执行这个命令估计的tempdb数据库需要的空间（并没有真的执行完整性检查）
PHYSICAL_ONLY	PHYSICAL_ONLY限制只对物理问题进行完整性检查，忽略逻辑检查
DATA_PURITY	这个参数用于升级数据库（SQL Server 2005以前的版本），这个参数指示DBCC CHECKTABLE检测不符合数据类型的列值（例如，bigint大小的值存储在整数值中）。在消除了所有升级数据库中的坏值之后，SQL Server会继续保持列值的完整性

这个技巧提供了几个使用这个命令的示例。在第一个示例中，对AdventureWorks数据库中的表Production.Product进行完整性检查：

```
DBCC CHECKTABLE ('Production.Product')
WITH ALL_ERRORMSGS
```

这个命令返回（结果基于你的环境可能会有不同）：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Production.Product的DBCC结果。
对象'Production.Product'的13页中有504行。
DBCC执行完毕。如果DBCC输出了错误信息，请与系统管理员联系。
```

在技巧的下一个示例中，返回检查Sales.SalesOrderDetail表估计所需要的tempdb空间。这样就可以提前了解到指定的CHECKTABLE操作是否需要比可用空间更多的空间：

```
DBCC CHECKTABLE ('Sales.SalesOrderDetail')
WITH ESTIMATEONLY
```

这个命令返回（由于结果部分基于本地环境，这个结果可能与你的不同）：

```
Estimated TEMPDB space needed for CHECKTABLES (KB)
897

DBCC执行完毕。如果DBCC输出了错误信息，请与系统管理员联系。
```

最后的示例对指定的索引执行DBCC CHECKTABLE，只检查物理错误（不是逻辑的）。但是，首先需要确定索引ID：

```
SELECT index_id
FROM   sys.indexes
WHERE   object_id = OBJECT_ID('Sales.SalesOrderDetail')
AND name = 'IX_SalesOrderDetail_ProductID'
```

这个查询返回：

```
index_id
3
```

678

然后，index_id将用到命令中：

```
DBCC CHECKTABLE ('Sales.SalesOrderDetail', 3)
WITH PHYSICAL_ONLY
```

这个命令返回：

```
DBCC执行完毕。如果DBCC输出了错误信息，请与系统管理员联系。
```

解析

在这个技巧中，第一个示例演示了如何检查单个表的完整性，如果存在错误就显示错误消息（而不是默认的最多返回200个消息）。第一个参数是要检查的表名称：

```
DBCC CHECKTABLE ('Production.Product')
```

第二个参数ALL_ERRORMSGS，指定返回发现的所有错误消息：

```
WITH ALL_ERRORMSGS
```

DBCC CHECKTABLE检查所有关于数据页链接、指针及分区中的行处于正确的分区，等等。

在第二个示例中，通过指定WITH ESTIMATEONLY参数为表Sales.SalesOrderDetail返回估计的tempdb所需的空间大小：

```
DBCC CHECKTABLE ('Sales.SalesOrderDetail')
WITH ESTIMATEONLY
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在最后的示例中，从系统目录视图sys.indexes中检索索引IX_SalesOrderDetail_ProductID的索引ID。在检索到索引ID之后，在使用了PHYSICAL_ONLY参数的DBCC CHECKTABLE命令中使用它，PHYSICAL_ONLY参数用来忽略对索引进行逻辑的完整性检查。

23.2.3 使用 DBCC CHECKCONSTRAINTS 检查表的完整性

DBCC CHECKCONSTRAINTS会报告在指定表和约束中发现的所有CHECK或外键约束的违反情况。这个命令允许返回违反约束的数据，从而可以修正违反的约束（但是如果使用了NOCHECK禁用了约束，那么这个命令也不会捕捉约束违反情况）。

语法如下：

```
DBCC CHECKCONSTRAINTS
[( 'table_name' | table_id | 'constraint_name' |
constraint_id )]
[ WITH
{ ALL_CONSTRAINTS | ALL_ERRORMSGS } [ , NO_INFOMSGS ] ]
```

表23-5描述了这个命令的参数。

679

表23-5 DBCC CHECKCONSTRAINTS参数

参 数	描 述
'table_name' \| table_id \| 'constraint_name' \| constraint_id	这个参数定义需要验证的表名称、表ID、约束名称或约束ID。如果没有指定特定对象，数据库中的所有对象都会被验证
ALL_CONSTRAINTS \| ALL_ERRORMSGS	当指定了ALL_CONSTRAINTS时，所有的约束（启用或禁用的）都会被检查。当指定了ALL_ERRORMSGS时，结果集中所有违反约束的行都会返回（而不是默认的最多返回200行）
NO_INFOMSGS	NO_INFOMSGS从DBCC输出中取消所有信息性消息

23

在这个技巧中，演示了在CHECK约束被违反后如何检查表的约束：

```
ALTER TABLE Production.WorkOrder NOCHECK CONSTRAINT CK_WorkOrder_EndDate
GO

-- 设置结束时间早于开始时间
UPDATE Production.WorkOrder
SET EndDate = '1/1/2001'
WHERE WorkOrderID = 1
GO

ALTER TABLE Production.WorkOrder CHECK CONSTRAINT CK_WorkOrder_EndDate
GO

DBCC CHECKCONSTRAINTS ('Production.WorkOrder')
```

这个命令返回如下结果：

Table	Constraint	Where
[Production].[WorkOrder]	[CK_WorkOrder_EndDate]	[StartDate] = '2001-07-04 00:00:00.000' AND [EndDate] = '2001-01-01 00:00:00.000'

解析

在这个技巧中，使用ALTER TABLE...NOCHECK CONSTRAINT，禁用了表Production.WorkOrder上名为

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

CK_WorkOrder的CHECK约束：

```
ALTER TABLE Production.WorkOrder NOCHECK CONSTRAINT CK_WorkOrder_EndDate
GO
```

它禁用了EndDate列值限定不得小于StartDate列的日期的约束。

在禁用了约束之后，更新了违反这个CHECK约束规则的行：

```
UPDATE Production.WorkOrder
SET EndDate = '1/1/2001'
WHERE WorkOrderID = 1
GO
```

680

然后重新启用约束：

```
ALTER TABLE Production.WorkOrder
CHECK CONSTRAINT CK_WorkOrder_EndDate
GO
```

然后对这个表执行DBCC CHECKCONSTRAINTS命令：

```
DBCC CHECKCONSTRAINTS('Production.WorkOrder')
```

这个命令返回表名、违反的约束以及违反约束的原因：

Table	Constraint	Where
[Production].[WorkOrder]	[CK_WorkOrder_EndDate]	[StartDate] = '2001-07-04 00:00:00.000' AND [EndDate] = '2001-01-01 00:00:00.000'

注解　不像其他的一些数据库完整性DBCC命令，DBCC CHECKCONSTRAINTS不会在DBCC CHECKDB中运行，因此如果需要标识数据库中数据约束的违反情况，必须作为单独的操作执行这个命令。

23.2.4　使用 DBCC CHECKCATALOG 检查系统表的一致性

DBCC CHECKCATALOG检查系统表内及之间的一致性（这是DBCC CHECKDB执行时自动包含的另外一个命令）。

语法如下：

```
DBCC CHECKCATALOG
[ ( 'database_name' | database_id | 0)]
    [ WITH NO_INFOMSGS ]
```

表23-6描述了这个命令的参数。

表23-6　DBCC CHECKCATALOG参数

参　数	描　述
'database_name' \| database_id \| 0	这个参数定义要检查错误的数据库名称或数据库ID。如果指定为0，则使用当前数据库
NO_INFOMSGS	NO_INFOMSGS从DBCC输出中取消所有信息性消息

在这个简短的技巧中，对整个AdventureWorks数据库进行系统表一致性检查：

681

```
DBCC CHECKCATALOG ('AdventureWorks')
```

这个命令返回如下内容（假定没有发现错误）：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DBCC执行完毕。如果DBCC输出了错误信息，请与系统管理员联系。
```

解析

在这个技巧中，检查了AdventureWorks数据库的系统目录数据。识别出错误时，会在命令输出中返回。如果发现错误，DBCC CHECKCATALOG没有修复选项可用，所以从最近完好的数据库备份中恢复可能是唯一的修复选择了。

像本章中描述的其他命令一样，当执行DBCC CHECKCATALOG时，在操作期间会创建一个内部数据库快照来维护事务一致性。如果因为某些原因不能创建数据库快照，在执行命令期间将产生排他数据库锁（这样会潜在地降低数据查询并发性）。

注解 CHECKCATALOG已经在DBCC CHECKDB命令中自动执行了，因此除非希望只检查系统表的一致性问题，没有必要分别执行它们。

23.3 索引维护

碎片是表修改自然会产生的副产品。当数据库内的数据被更新，索引的逻辑顺序（基于索引键）就会与数据页实际的物理顺序不同步。当数据页变得越来越无序，为了返回查询请求的结果就会需要更多的I/O操作。重建或重组织索引可以通过同步逻辑索引顺序和重组织物理数据页去匹配逻辑索引顺序，来对索引进行碎片整理。在下面两个技巧中，将演示两个可以用来对索引进行碎片整理的方法。

注解 请参看第28章学习如何显示索引碎片。这在只重建必要索引时是重要的。重建过程是很耗费资源的，如果碎片少，或查询主要是单独的查找而非范围扫描则影响最小。

23.3.1 重建索引

如果你使用过SQL Server早期的版本，可能会在本章寻找DBCC DBREINDEX或DBCC INDEXDEFRAG命令，它们分别用来重建索引和对索引进行碎片整理。DBCC DBREINDEX已不再建议使用，取而代之的是ALTER INDEX REBUILD命令。DBCC INDEXDEFRAG命令用来对索引进行碎片整理，同时允许进行数据访问——也不再建议使用，取代它的是ALTER INDEX REORGANIZE命令（会在后面的技巧中提到）。

重建索引有很多用途，最常用的是对表一直发生数据修改引发的碎片移除。当碎片增长时，查询会变慢。重建索引可以移除索引行的碎片并且释放出物理磁盘空间。682

有太多碎片的大型索引会降低查询速度。重建索引的频度依赖于数据库的容量大小、数据修改的多少、在表上活动的多少以及查询是否为典型的排序扫描或单个的查找。

用来重建索引的ALTER INDEX的语法如下：

```
ALTER INDEX { index_name | ALL }
    ON <object>
    { REBUILD
    [ [ WITH ( <rebuild_index_option> [ ,...n ] ) ]
    | [ PARTITION = partition_number
[ WITH ( <single_partition_rebuild_index_option>
                        [ ,...n ] )
                ]
            ]
        ]
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表23-7描述这个命令的参数。

表23-7　ALTER INDEX...REBUILD的参数

参　数	描　述
index_name \| ALL	这个参数定义要重建的索引名称。如果指定ALL，所有指定表或视图的索引将被重建
<object>	这个参数指定索引所在的表或视图的名称
<rebuild_index_option>	在重建过程中可以应用的一个或多个索引选项，包括：FILLFACTOR、PAD_INDEX、SORT_IN_TEMPDB、IGNORE_DUP_KEY、STATISTICS_NORECOMPUTE、ONLINE、ALLOW_ROW_LOCKS、ALLOW_PAGE_LOCKS、DATA_COMPRESSION和MAXDOP
partition_number	如果使用了分区索引，partition_number指定只对一个分区进行索引重建
<single_partiton_rebuild_index_option>	如果指定了重建索引分区，你会被限制使用在如下的WITH子句中使用的索引选项：SORT_IN_TEMPDB和MAXDOP（这两者都在rebuild_index_option中描述过）

这个技巧演示了ALTER INDEX REBUILD，它删除并重新创建了既有的索引。它演示了AdventureWorks数据库中重建索引的一些变化情况：

```
-- 重建指定索引

ALTER INDEX PK_ShipMethod_ShipMethodID
ON Purchasing.ShipMethod REBUILD

-- 重建指定表的所有索引

ALTER INDEX ALL
ON Purchasing.PurchaseOrderHeader REBUILD

-- 保持索引在查询时可用重建索引（需要企业版）

ALTER INDEX PK_ProductReview_ProductReviewID
ON Production.ProductReview REBUILD
WITH (ONLINE = ON)

-- 使用新填充因子并在tempdb中排序重建索引

ALTER INDEX PK_TransactionHistory_TransactionID
ON Production.TransactionHistory REBUILD
WITH (FILLFACTOR = 75,
SORT_IN_TEMPDB = ON)

-- 用启用压缩的页级别数据重建索引

ALTER INDEX PK_ShipMethod_ShipMethodID
ON Purchasing.ShipMethod REBUILD
WITH (DATA_COMPRESSION = PAGE)
```

683

解析

在这个技巧中，首先使用ALTER INDEX重建表Purchasing.ShipMethod的主键索引（重建聚集索引并不会引起表中任何非聚集索引的重建）：

```
ALTER INDEX PK_ShipMethod_ShipMethodID
ON Purchasing.ShipMethod REBUILD
```

在第二个示例中，使用了ALL关键字，它的意思是，不管是非聚集的还是聚集的，所有索引（要

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

记住，表中只允许存在一个聚集索引）都会被重建：

```
ALTER INDEX ALL
ON Purchasing.PurchaseOrderHeader REBUILD
```

在技巧中的第三个示例，联机重建索引，这意味着在重建索引PK_ProductReview_ProductReviewID时，用户查询可以继续访问这个索引的数据。

```
WITH (ONLINE = ON)
```

只有SQL Server企业版支持ONLINE选项，并且不能使用在XML索引、禁用的索引或分区索引上。并且使用了大型对象数据类型或是建立在临时表上的索引也不能享受这个选项带来的好处。

在第四个示例中，修改两个索引选项以应用到索引中——填充因子以及在tempdb中排列临时索引结果的指示：

```
WITH (FILLFACTOR = 75,
SORT_IN_TEMPDB = ON)
```

在最后一个示例中，使用压缩的页级别数据重建未压缩的索引

```
WITH (DATA_COMPRESSION = PAGE)
```

提示 可以通过查看sys.partitions的data_compression_desc字段来验证索引/分区是否压缩。 684

23.3.2 索引碎片整理

ALTER INDEX REORGANIZE减少了索引（聚集索引和非聚集索引）叶级别中的碎片，使数据库页的物理顺序与逻辑顺序匹配。在处理过程中，基于填充因子对索引进行压缩，从而释放出磁盘空间并且使索引变得更小。ALTER INDEX REORGANIZE是自动的联机操作，这意味着你可以在重组织过程中继续查询目标数据。

语法如下：

```
ALTER INDEX { index_name | ALL }
    ON <object>
    { REORGANIZE
        [ PARTITION = partition_number ]
        [ WITH ( LOB_COMPACTION = { ON | OFF } ) ]
    }
```

表23-8描述了这个命令的参数。

表23-8 ALTER INDEX...REORGANIZE的参数

参 数	描 述
index_name \| ALL	这个参数定义要重建的索引名称。如果指定ALL，表或视图中所有的索引都将被重建
<object>	这个参数指定要创建的索引所在的表或视图的名称
partition_number	如果使用了分区索引，partition_number指定了要重组织的分区
LOB_COMPACTION = { ON \| OFF}	当启用时，会对大型对象数据类型（varchar(max)、nvarchar(max)、varbinary(max)、xml、text、ntext和image数据）进行压缩

这一技巧演示了如何对单个索引，以及单个表中所有的索引进行碎片整理：

```
-- 重组织指定的索引
ALTER INDEX PK_TransactionHistory_TransactionID
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ON Production.TransactionHistory
REORGANIZE

-- 重组织表中所有的索引
-- 压缩大型对象数据类型
ALTER INDEX ALL
ON HumanResources.JobCandidate
REORGANIZE
WITH (LOB_COMPACTION=ON)
```

解析

在这一技巧的第一个示例中，重组织（碎片整理）了表Production.TransactionHistory的主键索引。它的语法与重建索引非常相似，只是将REBUILD替换为REORGANIZE关键字。

在第二个示例中，对HumanResources.Jobcandidate列中所有的索引进行了碎片整理（使用了关键字ALL）。并且使用了WITH子句，对大型对象数据类型列进行了压缩。

685

如果不能承受进行索引重建时将索引置为离线状态（并且如果因为没有运行SQL Server企业版，不能在ALTER INDEX REBUILD中使用ONLINE选项），就可以使用ALTER INDEX REORGANIZE。重组织始终是联机操作，这意味着尽管ALTER INDEX REORGANIZE操作可能比REBUILD处理速度要慢，但是它不会在重要的时间阻塞数据库的通信。

23.3.3　重建堆

在SQL Server 2008中，现在可以使用ALTER TABLE命令重建堆（没有聚集索引的表）。在以前的版本中，重建堆需要添加和移除临时聚集索引，或是执行数据迁移或表重建。

在这个示例中，将创建堆表（使用SELECT INTO），然后重建它：

```
USE AdventureWorks
GO

-- 创建基于另一个表的无索引的表
SELECT ShiftID, Name, StartTime, EndTime, ModifiedDate
INTO dbo.Heap_Shift
FROM HumanResources.Shift
```

可以通过查询sys.indexes来验证新表是否为堆：

```
SELECT type_desc
FROM sys.indexes
WHERE object_id = OBJECT_ID('Heap_Shift')
```

这个查询返回：

```
type_desc
HEAP
```

如果希望重建堆，可以执行如下的ALTER TABLE命令：

```
ALTER TABLE dbo.Heap_Shift REBUILD
```

解析

在这个技巧中，创建堆表，然后使用ALTER TABLE...REBUILD重建它。即使它没有聚集索引（堆），使用ALTER TABLE...REBUILD命令也可以重建表。如果表是分区表，这个命令也可以重建所有分区，并

686

重建存在的聚集索引。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第24章 维护数据库对象和对象依赖关系

本章包含一些用来维护数据库对象和查看对象依赖关系的技巧，你将看到用来完成以下操作的技巧。

- 修改用户创建的数据库对象名称。
- 修改对象架构。
- 通过新的SQL Server 2008的sys.sql_expression_dependencies视图和动态管理视图sys.dm_sql_referenced_entities及sys.dm_sql_referencing_entities来显示关于对象依赖关系的信息。

提示 在SQL Server 2008中，对象依赖关系跟踪更加健壮，使用它可以对跨数据库和跨服务器的依赖关系进行可靠的跟踪。现在可以通过对象名称而不是ID来跟踪依赖关系，支持在延迟名称解析的情况下跟踪依赖关系。

24.1 数据库对象维护

在下面的两个技巧中，将展示如何通过使用系统存储过程sp_rename修改用户创建的既有的数据库对象的名称，以及如何通过使用ALTER SCHEMA将存在于一个架构中的既有的对象转移到不同的架构中去。

24.1.1 修改用户创建的数据库对象的名称

这个技巧展示了如何使用系统存储过程sp_rename重命名对象。使用这个存储过程可以重命名表列、索引、表、约束以及其他数据库对象。

sp_rename的语法如下：

```
sp_rename [ @objname = ] 'object_name' , [ @newname = ] 'new_name'
    [ , [ @objtype = ] 'object_type' ]
```

表24-1中描述了这个系统存储过程的参数。

表24-1 sp_rename参数

参 数	描 述
object_name	要重命名的对象名称
new_name	对象的新名称
object_type	要重命名的对象类型：column、database、index、object和userdatatype

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个技巧展示了如何重命名表、列、索引：

```
USE AdventureWorks
GO

-- 增加示例对象

CREATE TABLE HumanResources.InsuranceProvider
(InsuranceProviderID int NOT NULL,
InsuranceProviderNM varchar(50) NOT NULL
)
GO

CREATE INDEX ni_InsuranceProvider_InsuranceProviderID
ON HumanResources.InsuranceProvider (InsuranceProviderID)

-- 重命名这个表
EXEC sys.sp_rename 'HumanResources.InsuranceProvider',
            'Provider',
            'Object'

-- 重命名列
EXEC sys.sp_rename 'HumanResources.Provider.InsuranceProviderID',
            'ProviderID',
            'Column'

-- 重命名主键约束
EXEC sys.sp_rename 'HumanResources.Provider.ni_InsuranceProvider_
InsuranceProviderID',
           'ni_Provider_ProviderID',
           'Index'
```

每个sp_rename执行后返回如下信息：

```
警告: 更改对象名的任一部分都可能会破坏脚本和存储过程。
```

解析

这个技巧首先创建了一个名为HumanResources.InsuranceProvider的新表，这个表中有一个索引名为InsuranceProviderID。完成之后，使用系统存储过程sp_rename重命名表：

```
EXEC sys.sp_rename 'HumanResources.InsuranceProvider',
            'Provider',
            'Object'
```

688

注意第一个参数使用了完全限定的对象名（schema.table_name），但是第二个参数只是使用新的table_name。第三个参数指定对象类型为object。

下一步，使用sp_rename修改列名称：

```
EXEC sys.sp_rename 'HumanResources.Provider.InsuranceProviderID',
            'ProviderID',
            'Column'
```

第一个参数用schema.table_name.column_name指定了要重命名的列，第二个参数是这个列的新名称。第三个参数指定对象类型为column。

在这个技巧的最后，重命名了索引：

```
EXEC sys.sp_rename 'HumanResources.Provider.ni_InsuranceProvider_
InsuranceProviderID',
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
'ni_Provider_ProviderID',
'Index'
```

第一个参数使用了schema.table_name.index_name参数。第二个参数指定了新索引的名称。第三个参数指定了对象类型为index。

这个技巧中返回了关于“更改对象名的任一部分都可能会破坏脚本和存储过程”的警告。在真实的工作情形中，在重命名对象之前，你也希望ALTER所有视图、存储过程、函数或其他引用了这个源对象名称的可编程对象。为了找到哪个对象引用了某个对象，稍后请看24.2.1节中的技巧。

24.1.2 修改对象的架构

在SQL Server 2000中，在引入架构的概念之前，使用系统存储过程sp_changeobjectowner修改对象的拥有者。现在在SQL Server 2005和SQL Server 2008中，用户（拥有者）和架构是分离的，并且使用ALTER SCHEMA命令修改对象的架构。

语法如下：

```
ALTER SCHEMA schema_name TRANSFER object_name
```

命令使用了两个参数：第一个是希望将对象转移到的架构名称，第二个是希望转移的对象名称。

这个技巧展示了将表从架构Sales转移到架构HumanResources：

```
Use AdventureWorks
GO

CREATE TABLE Sales.TerminationReason
(TerminationReasonID int NOT NULL PRIMARY KEY,
TerminationReasonDESC varchar(100) NOT NULL)
GO

ALTER SCHEMA HumanResources TRANSFER Sales.TerminationReason
GO
```

解析

在这个技巧中，在架构Sales中创建了一个新表。一个对象不再被一个特定的用户所拥有，而是包含在架构中。创建表之后，使用ALTER SCHEMA TRANSFER命令将表转移到架构HumanResources中。

警告 原授予schema.object的权限将在转移后删除（例如USER1的SELECT权限）。如果这些权限需要在新架构中保留，确保要在使用ALTER SCHEMA之前实施这些权限授予脚本。

24.2 对象依赖关系

SQL Server 2008引入了更可靠的跟踪对象依赖关系的方法，后面几个技巧会演示。

警告 在这个版本的SQL Server中，不再建议使用sp_depends系统存储过程和sys.sql_dependencies目录视图，而是使用在本章中演示的视图来跟踪对象依赖关系。

24.2.1 标识对象依赖关系

SQL Server 2008提供了新的标识数据库中、跨数据库和跨服务器（使用链接服务器4部分名称）的对象依赖关系的方法。这个技巧将演示使用sys.sql_expression_dependencies对象目录视图在几种

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

情况下标识依赖关系。

为了演示这个功能，首先创建了两个新数据库以及一些新对象：

```
USE master
GO

-- 创建两个新数据库
CREATE DATABASE TSQLRecipe_A
GO

CREATE DATABASE TSQLRecipe_B
GO

-- 在第一个数据库中创建新表
USE TSQLRecipe_A
GO

CREATE TABLE dbo.Book
  (BookID int NOT NULL PRIMARY KEY,
   BookNM varchar(50) NOT NULL)
GO

-- 创建引用第二个数据库中对象的存储过程
USE TSQLRecipe_B
GO

CREATE PROCEDURE dbo.usp_SEL_Book
AS

SELECT BookID, BookNM
FROM TSQLRecipe_A.dbo.Book
GO
```

690

上面还创建了引用另一数据库中表的存储过程。现在如果要查看所有存储过程依赖的对象，可以执行如下的对目录视图sys.sql_expression_dependencies的查询操作（我会在“解析”部分详细说明列的含义）：

```
SELECT referenced_server_name, referenced_database_name,
   referenced_schema_name, referenced_entity_name, is_caller_dependent
FROM sys.sql_expression_dependencies
WHERE OBJECT_NAME(referencing_id) = 'usp_SEL_Book'
```

这个查询返回一行（简略重排格式）：

```
referenced_server_name      NULL
referenced_database_name    TSQLRecipe_A
referenced_schema_name      dbo
referenced_entity_name      Book
is_caller_dependent         0
```

现在假定创建了另一个引用尚不存在对象的存储过程（这对存储过程来说是可以的，也是一个常用的做法）。例如：

```
-- 创建引用第二个数据库中对象的存储过程
USE TSQLRecipe_B
GO

CREATE PROCEDURE dbo.usp_SEL_Contract
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
AS

SELECT ContractID, ContractNM
FROM TSQLRecipe_A.dbo.Contract
GO
```

在SQL Server早期的版本中，不能跟踪不存在对象的依赖关系，在SQL Server 2008中，这个行为现在是允许的。可以执行如下查询来检查usp_SEL_contract的依赖关系：

```
USE TSQLRecipe_B
GO

SELECT referenced_server_name, referenced_database_name,
   referenced_schema_name, referenced_entity_name, is_caller_dependent
FROM sys.sql_expression_dependencies
WHERE OBJECT_NAME(referencing_id) = 'usp_SEL_Contract'
```

这个查询返回一行（简略重排格式）：

```
referenced_server_name      NULL
referenced_database_name    TSQLRecipe_A
referenced_schema_name      dbo
referenced_entity_name      Contract
is_caller_dependent         0
```

691

即使不存在TSQLRecipe_A.dbo.Contract对象，存储过程的引用和被引用的表之间的依赖关系仍然表示出来了。

解析

这个技巧演示如何使用sys.sql_expression_dependencies目录视图来确定对象的依赖关系。在SELECT子句中，引用了5列。第1列referenced_server_name引用了4部分链接服务器名称（当存在时）。如果使用了3部分数据库、架构和对象，则referenced_database_name返回非空值。如果在引用的模块中使用了两部分架构和对象名称，则referenced_schema_name可用。referenced_entity_name是引用的对象名称。最后，is_caller_dependent列表示对象引用是否依赖执行模块的人。例如，如果没有完全限定对象名称，而且名为T1的对象存在于两个不同的架构中，实际引用的对象依赖于调用模块的人和执行的上下文。

24

24.2.2 标识引用和被引用的实体

SQL Server 2008引入了两个用来标识被引用对象和引用对象的新动态管理视图，正如这个技巧演示的。为sys.dm_sql_referenced_entities动态管理视图提供引用对象的名称，返回被引用对象的结果集。为sys.dm_sql_referencing_entities动态管理视图提供被引用对象的名称，返回引用对象的结果集。

在第一个示例中，将创建两次被引用的表，一次引用来自于存储过程，另一次引用来自于视图：

```
USE TSQLRecipe_A
GO

CREATE TABLE dbo.BookPublisher
   (BookPublisherID int NOT NULL PRIMARY KEY,
    BookPublisherNM varchar(30) NOT NULL)
GO

CREATE VIEW dbo.vw_BookPublisher
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
AS

SELECT BookPublisherID, BookPublisherNM
FROM dbo.BookPublisher
GO

CREATE PROCEDURE dbo.usp_INS_BookPublisher
   @BookPublisherNM varchar(30)
AS

INSERT dbo.BookPublisher
(BookPublisherNM)
VALUES (@BookPublisherNM)

GO
```

692 下一步，使用sys.dm_sql_referenced_entities动态管理视图显示视图本身引用的所有对象：

```
SELECT referenced_entity_name, referenced_minor_name
FROM sys.dm_sql_referenced_entities ('dbo.vw_BookPublisher', 'OBJECT')
```

这个查询返回：

```
referenced_entity_name      referenced_minor_name
BookPublisher               NULL
BookPublisher               BookPublisherID
BookPublisher               BookPublisherNM
```

注意这个函数显示视图引用的表的一行数据，还有两行关于视图引用的列的数据。我将在“解析”部分中讨论参数。

下一步，将查询sys.dm_sql_referencing_entities动态管理视图来验定所有引用了dbo.BookPublisher表的对象（动态管理视图的名称相似，因此注意我使用了“引用”而不是“被引用”）：

```
SELECT referencing_schema_name, referencing_entity_name
FROM sys.dm_sql_referencing_entities ('dbo.BookPublisher', 'OBJECT')
```

这个查询返回：

```
referencing_schema_name     referencing_entity_name
dbo                         usp_INS_BookPublisher
dbo                         vw_BookPublisher
```

解析

这个技巧演示了两个标识对象依赖关系的方法。创建了表，然后在视图和存储过程中引用它。sys.dm_sql_referenced_entities动态管理视图用来返回指定对象中引用实体的列表。

```
sys.dm_sql_referenced_entities ('dbo.vw_BookPublisher', 'OBJECT')
```

第一个参数是引用了其他对象的对象名称。第二个参数指定列表实体的类型。可以从OBJECT、DATABASE_DDL_TRIGGER和SERVER_DDL_TRIGGER中选择。我选择了OBJECT，结果为被引用表的名称和在视图内SELECT子句中指定的列。

技巧中第二个示例演示了显示所有对指定对象的引用，在这个例子中是在技巧开始时创建的表：

```
FROM sys.dm_sql_referencing_entities ('dbo.BookPublisher', 'OBJECT')
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第一个参数是要标识的引用到的对象名称。第二个参数指定了要标识的对象的类。可从OBJECT、TYPE、XML_SCHEMA_COLLECTION和PARTITION_FUNCTION中选择。我选择了OBJECT，结果为引用第一个参数中指定表的视图和存储过程的列表。

693

24.2.3 查看对象定义

当标识出依赖一个你想修改的模块的对象时，可以使用函数OBJECT_DEFINITION看一下它的定义。这个函数可以用来返回用户定义和基于系统的约束、默认值、存储过程、函数、角色、架构级DML和DDL触发器及视图的T-SQL定义。

语法如下：

```
OBJECT_DEFINITION ( object_id )
```

这个命令只有一个参数，是对象ID，它是唯一的对象标识符（所有的对象标识符唯一地标识数据库中的数据库对象）。

在这个示例技巧中，返回数据库AdventureWorks的用户定义函数的T-SQL定义，并在函数OBJECT_DEFINITION中使用函数OBJECT_ID来得到用户定义函数的标识符：

```
USE AdventureWorks
GO

SELECT OBJECT_DEFINITION
(OBJECT_ID('dbo.ufnGetAccountingEndDate'))
GO
```

这个查询返回T-SQL定义：

```
CREATE FUNCTION [dbo].[ufnGetAccountingEndDate]()
RETURNS [datetime]
AS
BEGIN
    RETURN DATEADD(millisecond, -2, CONVERT(datetime, '20040701', 112));
END;
```

24

如果想知道微软如何为系统对象编程，也可以使用OBJECT_DEFINITION去窥视它们的T-SQL定义。在这个示例中，显示了系统存储过程sp_depends的代码：

```
USE AdventureWorks
GO

SELECT OBJECT_DEFINITION(OBJECT_ID('sys.sp_depends'))
GO
```

这个查询返回如下部分结果：

```
create procedure sys.sp_depends
--- 1996/08/09 16:51 @objname nvarchar(776)
...
select @dbname = parsename(@objname,3)

if @dbname is not null and @dbname <> db_name()
begin raiserror(15250,-1,-1)
return (1)
end
...
```

694

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个技巧中展示了使用OBJECT_DEFINITION返回用户定义函数和系统存储过程的T-SQL代码。在两个示例中，函数OBJECT_ID被嵌套在函数中，以传递作为参数的对象标识符ID：

```
OBJECT_DEFINITION
(OBJECT_ID('dbo.ufnGetAccountingEndDate'))
```

对象名称是使用schema.object_name格式完全限定的。

两个示例都返回数据库对象的T-SQL代码定义。如果那些对象是加密的，只会得到为NULL的结果集。

695

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第25章 数据库镜像

数据库镜像（database mirroring）功能为用户级数据库提供了高可用性。这里提到的高可用性是指对最终用户来说SQL Server数据库只有很少或没有非计划的停机时间，其他时间它处于可用状态。数据库镜像通过将一个服务器上的主（主体）数据库和另一个服务器上的数据库副本同步，允许数据库冗余。第二个副本可以用作热备用服务器，使用它可以在任何原因下需要将主副本离线时做一个快速的故障转移。不像故障转移集群，数据库镜像不需要像共享磁盘阵列或SAN那样昂贵的硬件。最低限度下，所需要的就是在同一网络中的两个SQL Server实例。

警告 不像故障转移集群，数据库镜像只能处理用户级数据库，不能镜像系统数据库（master、msdb、tempdb）。

在本章中，将介绍如何安装、配置、监控以及删除数据库镜像。在下一节开始数据库镜像具体内容前，先讲一下在SQL Server高可用性选项上下文中的数据库镜像。

提示 尽管数据库镜像的核心功能没有变，SQL Server 2008还是引入了数据流压缩和自动页恢复功能。这些提升不需要变更已存在的语法。

25.1 上下文中的数据库镜像

数据库镜像是作为SQL Server中的一组高可用性技术之一提供的。

- 故障转移集群（failover clustering）。在SQL Server的上一个版本中就是可用的，故障转移集群可以维护SQL Server实例级别的高可用性，使用两个或更多连接到共享磁盘的节点。当你将SQL Server实例安装到故障转移集群时，用户数据库和系统数据库文件被安装到共享磁盘，并且常规二进制安装文件被写入参与集群中的所有节点（服务器）。每次由一个物理节点“控制”SQL Server实例，如果这个节点发生的故障导致节点不可用，集群上的另一个节点可以接管SQL Server实例的服务。依赖这个SQL Server实例的设置，将故障从一个服务器转移到另一个服务器可能只需几秒钟。
- 日志传送（log shipping）。在以前的版本中也是可用的，日志传送与数据库镜像非常相似。日志传送在数据库级别启用高可用性，包含在一个SQL Server实例中的主在线数据库，以及持续在另一个SQL Server实例上进行恢复的数据库。当在主数据库执行了事务日志备份，事务日志备份被复制到另一个SQL Server实例中，并且持续地应用到这个数据库副本中。在主数据库或它所在的服务器出现故障的情形下，可以将另一个数据库副本通过应用最后传送过来的事务

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

日志备份设为联机。

- 复制（replication）。在以前的版本中也是可用的。复制可以将数据库和对象定义（表、视图等）移动到在一个或多个SQL Server实例中的另一个数据库副本上。根据选择的复制类型，可以通过具体的时间表去推动数据变化，在数据变化时进行数据迁移，或者在多个数据源中同步数据变化。复制以较小的形式提供高可用性，它关注特定的对象和数据，但是不允许自动传送所有的数据库对象类型。它的意思是不能基于它为数据库生成一个相同的副本（有时候数据库镜像是可以做到的）。

数据库镜像和这些高可用性选项之间存在如下一些差别。

- 数据库镜像不需要故障转移集群需要的共享磁盘或专门的硬件。故障转移集群保护整个SQL Server实例，但是数据库镜像只在用户数据库的范围允许高可用性。系统数据库不可以被镜像。
- 不像日志传送，安装数据库镜像可以完全通过T-SQL来完成。日志传送则需要手动配置并且是包含在温备用解决方案中的。数据库镜像被集成到数据库引擎并且允许用作热数据库备用，允许瞬间进行故障转移。
- 复制可以推或拉特定的数据库对象，但是不可以去拉所有的数据库对象。可是数据库镜像会创建精确的数据库副本。

注解　热备用服务器是指频繁地接收来自生产服务器的更新，并且可以在生产服务器发生故障的情况下马上变为可用状态的服务器。温备用服务器是指接收更新的服务器，但在接管生产服务器上的故障事件之前可能需要调整或几分钟切换。

在下一部分将讨论数据库镜像的体系结构。

25.2　数据库镜像体系结构

数据库镜像提供了主体服务器角色、镜像服务器角色和一个可选的见证服务器（参见图25-1）。在主体服务器中的数据库是实际被使用的，并且因为事务应用在主体服务器的数据库上，它们也被提交到镜像服务器的数据库上。镜像服务器数据库被置为恢复状态，在主体的副本上接收更新，但在主数据库仍然可用的情况下是不能使用镜像服务器的。

698

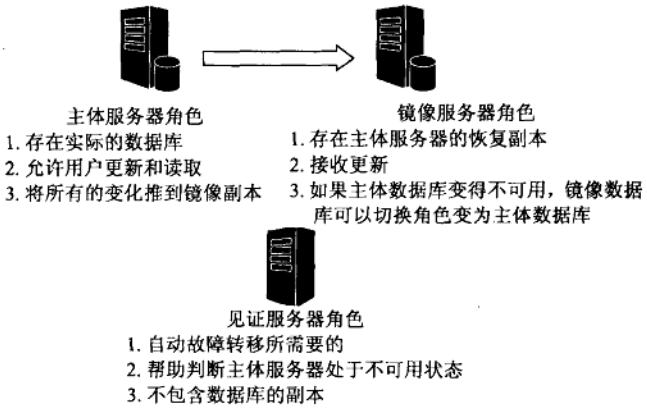


图25-1　数据库镜像基本体系结构

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果主体数据库发生了问题，然后引起了故障，镜像服务器可以转换为主数据库角色。当另一个数据库（源主体）回到联机状态，以前的转换到镜像服务器数据库角色的主数据库，接收来自主体服务器的事务。

从主体数据库到镜像数据库的故障转移可以通过手动或自动的方式被初始化，这取决于数据库的镜像模式。如果需要自动故障转移，必须将第三个服务器作为见证服务器加入到镜像会话中。见证服务器（witness server）监视主体服务器和镜像服务器。在数据库镜像会话中包括3个服务器（主体、镜像和见证），两个或3个相连接的服务器就可以生成关于是否进行自动故障转移的决策，称为仲裁（quorum）。

数据库镜像会话可以运行在同步或异步的模式下。当处于同步模式时，写入主体服务器数据库的事务在任何包含的事务可以提交之前，也必须写入镜像服务器数据库。这个选项保证了数据的冗余，但是作为代价，它也带来了潜在的性能降低。

异步模式在真正将事务写入镜像服务器数据库之前，允许主体数据库镜像会话进行事务提交。这个选项允许主体数据库更快地完成事务，但在更新可以被映射到镜像数据库之前，如果主体服务器发生了故障，就会引发丢失事务的危险。

提示 SQL Server 2008企业版提供在主体和镜像数据库中发生页损坏时自动数据页修复的功能。例如，当主体数据库中发生页损坏时，就会尝试从镜像数据库中检索未损坏的页。如果有必要，使用页的镜像副本来覆盖主体数据库中的坏页。当在镜像数据库中而不在主体数据库中发生页损坏也是可以修复的。

699

关于应用程序和客户端到主体数据库的连通性，SQL Server保留了在发生故障的情况下，允许.NET应用程序重定向的元数据。特别地，可以在.NET节点中使用SQL Native Client去连接镜像数据库，并且节点可以被配置知道镜像数据库的位置。可以在连接字符串中同时指定主体和镜像SQL Server实例来使用SQL Native Client，当主数据库不可用时，使用它可以使应用程序连接透明地重定向到新的活动主体服务器。

25.3 设置数据库镜像

25

在本章中，将通过一系列的小技巧展示一个场景，非常像第20章中所做的。在这个场景中，在BookStore数据库上设置一个数据库镜像会话。一个SQL Server实例将存在于主体数据库，另一个存在于镜像数据库，第三个作为见证服务器（不需要数据库）。

随后是启用数据库镜像的一般步骤。

(1) 创建端点。创建镜像端点，它允许SQL Server实例（主体、镜像、见证）之间互相通信。可以选择使用哪种身证验证方式，并且将在随后的技巧中讨论这个问题。

(2) 创建数据库镜像副本。在操作它之前，需要确认主体数据库被设为FULL恢复模式，因为为了传播主体数据库的修改，事务日志备份会从主体数据库应用到镜像数据库。为了创建镜像数据库副本，要对主体数据库做完全数据库备份，然后使用WITH NORECOVERY（这个选项将数据库置为接收附加事务日志还原的状态）将它还原到镜像SQL Server实例。在做完主体的完全数据库备份后，也必须要做事务日志备份，然后把它还原到数据库镜像副本。

(3) 初始化数据库镜像会话。这是最后一步，使用ALTER DATABASE指定每个数据库的角色。这个命令告诉SQL Server哪一个SQL Server端点连接到伙伴，哪一个连接到见证。伙伴服务器（partner database）

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

拥有主体和/或镜像数据库，并且如果主体数据库变为不可用也可以改变角色。

这个场景中的第一个技巧将展示如何创建镜像端点，镜像端点用来定义哪个SQL Server实例参与到数据库镜像会话中的哪一个活动。

25.3.1　创建镜像端点

为了建立镜像会话，参与的服务器必须能在它们自己专用的TCP端口上互相通信。这些端点的端口将被指定去监听镜像的消息和操作。

为了准备建立新的数据库镜像会话，首先在镜像服务器上创建端点，随后是主数据库，然后是可选的见证服务器（如果你希望自动进行故障转移就指定它）。

CREATE ENDPOINT命令用来创建镜像端点。回顾一下本书中前面的章节，CREATE ENDPOINT也用作创
700 建HTTP端点，并且启用Service Broker跨服务器通信。将它应用到数据库镜像中的语法如下：

```
CREATE ENDPOINT endPointName [ AUTHORIZATION login ]
STATE = { STARTED | STOPPED | DISABLED }
AS TCP (LISTENER_PORT = listenerPort )
FOR DATABASE_MIRRORING (
   [ AUTHENTICATION = {
            WINDOWS [ { NTLM | KERBEROS | NEGOTIATE } ]
      | CERTIFICATE certificate_name
   } ]
  [ [ , ] ENCRYPTION = { DISABLED |SUPPORTED | REQUIRED }
     [ ALGORITHM { RC4 | AES | AES RC4 | RC4 AES } ]
]
  [,] ROLE = { WITNESS | PARTNER | ALL }
)
```

表25-1描述了这个命令的参数。

表25-1　CREATE ENDPOINT...FOR DATABASE MIRRORING参数

参　数	描　述
endPointName	这个参数定义新服务器端点名称
login	这个选项指定端点所在的SQL Server或Windows的登录账户。如果没有指定它，默认的拥有者是新端点的创建者
STATE = { STARTED \| STOPPED \| DISABLED }	这个参数定义了端点创建时的状态。STARTED意思是端点将立即启动。DISABLED意思是端点将不监听或响应请求。STOPPED意思是端点监听请求，但向调用者返回错误
listenerPort	这个参数指定镜像会话监听进入的通信的自由TCP端口
WINDOWS [{ NTLM \| KERBEROS \| NEGOTIATE}]	这个选项指定了端点连接的身份验证方式，使用NTLM、KERBEROS或NEGOTIATE（NEGOTIATE允许端点使用Windows协商协议在NTLM和KERBEROS之间进行选择）。如果未指定，NEGOTIATE是默认身份验证选项
CERTIFICATE certificate_name	CERTIFICATE选项允许使用证书进行身份验证，需要调用端点时
ENCRYPTION = { DISABLED \| SUPPORTED \| REQUIRED }	这个选项在镜像过程中应用了加密。当选择DISABLED时，镜像会话中传送的数据是不加密的。当选择了SUPPORTED时，如果互相通信的端点都支持加密，就使用加密（否则就不加密）。REQUIRED指定通信端点必须支持加密
701 ALGORITHM { RC4 \| AES \| AES RC4 \| RC4 AES }	这个选项指定在加密数据传输时使用的加密算法
WITNESS \| PARTNER \| ALL	这些选项指定数据库镜像服务器角色。当指定PARTNER时，创建的端点可以用来进行主的或镜像会话通信。如果选择WITNESS，这个端点用在镜像会话中的见证角色上。ALL会话允许端点用在主、镜像和见证镜像会话中

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在开始这个技巧之前，首先需要讨论一下3个SQL Server实例互相通信所需要的身份验证选项。首先，只要每个SQL Server实例都运行在同一域服务账号，并且如果使用了WINDOWS选项创建端点，SQL Server实例将自动在数据库镜像会话中互相访问。但是，如果这些SQL Server实例没有运行在同一域服务账号之下，则需要在所有参与的SQL Server实例的远程SQL Server实例中创建Windows登录。例如，在主体数据库上的SQL Server实例使用了一个启动服务账号［JOEPROD\SQLAdmin］。假设镜像数据库副本上的SQL Server实例使用了启动服务账号［JOEPROD\Node2Admin］。为了允许镜像SQL Server访问主体SQL Server，［JOEPROD\Node2Admin］必须添加到主体数据库。例如：

```
USE master
GO
CREATE LOGIN [JOEPROD\Node2Admin]
FROM WINDOWS
GO
```

为了允许访问主体和见证SQL Server实例，必须在镜像SQL Server实例上做相同的操作。如果这些新账号也在其他SQL Server服务器的Windows管理员组中，那些登录将自动访问并连接到数据库镜像端点。但是，如果它们不是这个组的成员，必须显式地授权访问到端点的远程登录。例如：

```
GRANT CONNECT ON ENDPOINT::JOEPROD_Mirror
TO [JOEPROD\Node2Admin]
GO
```

在本章的场景中，将使用运行在同一Windows服务账号下的3个SQL Server实例。在这个示例的第一部分中，在保存镜像副本的数据库的SQL Server实例上创建一个新端点：

```
-- 在镜像SQL Server实例上创建端点

CREATE ENDPOINT JOEPROD_Mirror
    STATE = STARTED
    AS TCP ( LISTENER_PORT = 5022 )
    FOR DATABASE_MIRRORING (
    AUTHENTICATION = WINDOWS NEGOTIATE,
    ENCRYPTION = SUPPORTED,
    ROLE=PARTNER)
GO
```

702

下一步在全体数据库的SQL Server实例上创建新端点：

25

```
-- 在主SQL Server实例上创建端点

CREATE ENDPOINT JOEPROD_Mirror
    STATE = STARTED
    AS TCP ( LISTENER_PORT = 5022 )
    FOR DATABASE_MIRRORING (
       AUTHENTICATION = WINDOWS NEGOTIATE,
       ENCRYPTION = SUPPORTED,
       ROLE=PARTNER)
GO
```

第三步，在镜像数据库会话中作为见证服务器的SQL Server实例上创建新端点：

```
-- 在见证SQL Server实例上创建端点

CREATE ENDPOINT JOEPROD_Witness
    STATE = STARTED
    AS TCP ( LISTENER_PORT = 5022 )
    FOR DATABASE_MIRRORING (
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        AUTHENTICATION = WINDOWS NEGOTIATE,
        ENCRYPTION = SUPPORTED,
        ROLE=WITNESS)
GO
```

在创建端点后，可以通过查询系统目录视图sys.database_mirroring_endpoints验证端点设置。

在最终放置了数据库镜像副本的SQL Server实例中，随后的查询确认了端点的名称、状态（意思是，是否启动了）以及它的镜像角色：

```
SELECT name, state_desc, role_desc
FROM sys.database_mirroring_endpoints
```

这个查询返回：

```
name                 state_desc       role_desc
JOEPROD_Mirror       STARTED          ARTNER
```

然后在主体数据库的SQL Server实例中执行下面这个查询：

```
SELECT name, state_desc, role_desc
FROM sys.database_mirroring_endpoints
```

这个查询返回：

```
name                 state_desc       role_desc
JOEPROD_Mirror       STARTED          PARTNER
```

703

下一步，查询假定需要见证角色的SQL Server实例：

```
SELECT name, state_desc, role_desc
FROM sys.database_mirroring_endpoints
```

这个查询返回：

```
name                 state_desc       role_desc
JOEPROD_Witness      STARTED          WITNESS
```

解析

在建立数据库镜像会话之前，必须在参与的SQL Server实例上添加端点。这些端点使用TCP/IP协议在指定的端口上监听。

在这个技巧中，首先在镜像SQL Server实例上创建了一个名为JOEPROD_Mirror的端点。

```
CREATE ENDPOINT JOEPROD_Mirror
```

初始的状态设为STARTED，意思是创建的端点处于可以马上使用的状态：

```
STATE = STARTED
```

TCP监听端口被设为5022。这是端点用来监听数据库镜像通信的端口：

```
AS TCP ( LISTENER_PORT = 5022 )
```

端口号的选择是任意的，但一定要确认端口是可用的。如果SQL Server实例通过防火墙进行通信，为了可以通信，必须打开防火墙指定的镜像端口。

选择了WINDOWS NEGOTIATE身证验证方式，意思是在参与的SQL Server实例之间的通信使用了Windows的身证验证方式：

```
FOR DATABASE_MIRRORING (
   AUTHENTICATION = WINDOWS NEGOTIATE,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

指定了SUPPORTED加密方式，意思是如果互相通信的会话支持加密，就在数据传输中进行加密：

```
ENCRYPTION = SUPPORTED,
```

镜像服务器的ROLE是PARTNER，意思是端点可以在主体数据库或镜像服务器上使用：

```
ROLE=PARTNER)
```

用完全相同的操作，在主体SQL Server实例上执行CREATE ENDPOINT（再一次使用PARTNER），然后以WITNESS的角色在见证SQL Server实例上执行。

最后，我查询了系统目录视图sys.database_mirroring_endpoints，它包含了在每个SQL Server实例上可能存在的任何数据库镜像端点的信息。（在这个场景中，每个SQL Server实例上只有一个端点。）

704

25.3.2 备份和还原主体数据库

在创建数据库镜像会话过程中，一旦创建了端点，下一步就是为主体数据库创建数据库备份，然后将它还原到镜像SQL Server实例上。在还原了完全数据库备份之后，要进行事务日志备份，并将它应用到数据库镜像副本上。

注解 本章展示了BACKUP和RECOVERY技术。这些技术会在第29章详细描述。

在备份数据库之前，为了使用数据库镜像，数据库需要设为FULL恢复模式。在这个示例中，展示了在主体数据库SQL Server实例上进行这个操作：

```
-- 这是在主体数据库SQL Server实例上执行
USE master
GO

IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'BookStore')

CREATE DATABASE BookStore
GO

-- 确认数据库使用了FULL恢复模式

ALTER DATABASE BookStore
SET RECOVERY FULL
GO
```

下一步，执行完整数据库备份：

```
-- 备份BookStore DATABASE

BACKUP DATABASE BookStore
TO DISK =
'C:\Apress\Recipes\Mirror\principalbackup_BookStore.bak'
WITH INIT
```

完成了主SQL Server实例上的数据库备份之后，手动复制.bak文件到镜像SQL Server实例，它使用了NORECOVERY进行还原。NORECOVERY模式让数据库处于允许附加事务日志的状态：

```
RESTORE DATABASE BookStore
FROM  DISK = 'C:\Apress\Recipes\Mirror\principalbackup_BookStore.bak'
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
WITH MOVE 'BookStore' TO 'C:\Apress\Recipes\Mirror\BookStore.mdf',
     MOVE 'BookStore_log' TO 'C:\Apress\Recipes\Mirror\BookStore_log.ldf',
     NORECOVERY
GO
```

705

要牢记，为了数据库镜像可以工作，还原的数据库必须使用与主体数据库一样的名称。

如果有任何事务日志备份发生在主体SQL Server实例上执行了完全备份之后，且在镜像服务器上执行还原之前，则必须将这些事务日志备份（使用RESTORE）应用到镜像服务器数据库。在启用镜像之前，也必须在主体数据库上执行事务日志备份，然后将它还原到镜像副本上。

这个示例展示了在主体数据库上备份事务日志：

```
BACKUP LOG BookStore
TO DISK =
'C:\Apress\Recipes\Mirror\BookStore_tlog.trn'
WITH INIT
```

在主SQL Server实例上完成了事务日志备份之后，将.trn文件手动复制到镜像SQL Server实例上，并且在还原时使用NORECOVERY：

```
RESTORE LOG BookStore
FROM DISK = 'C:\Apress\Recipes\Mirror\BookStore_tlog.trn'
WITH FILE = 1, NORECOVERY
```

提示 还原用户数据库不需要进行SQL或Windows登录到包含镜像数据库的服务器。任何映射到主体数据库的数据库用户上的SQL或Windows登录也应该创建在镜像SQL Server实例上。当镜像数据库必须切换为主体角色时，这些登录应该在故障出现前就准备就绪。如果镜像数据库的SQL Server实例上的登录不存在，镜像数据库中的数据库用户将被孤立（将无法访问且没有任何关联登录名的数据库用户）。

解析

在这个技巧中，为了可以参与到数据库镜像会话中，主体数据库首先被修改为FULL恢复模式：

```
ALTER DATABASE BookStore
SET RECOVERY FULL
```

然后，对BookStore数据库进行完全数据库备份。使用INIT选项将刚才完成的完全备份整个覆盖到数据库文件上（在这个示例中，在指定的文件上已经存在了一个较老的备份）：

```
BACKUP DATABASE BookStore
TO DISK =
'C:\Apress\Recipes\Mirror\principalbackup_BookStore.bak'
WITH INIT
```

然后将备份文件手动复制到第二个位于数据库镜像副本中的SQL Server实例上。接着通过使用MOVE和NORECOVERY选项还原了一个新数据库。当希望重新定位数据库文件还原至何处时，应该使用MOVE选项，与它相对的是，当创建了原始备份时，备份是如何保存的：

```
RESTORE DATABASE BookStore
FROM DISK = 'C:\Apress\Recipes\Mirror\principalbackup_BookStore.bak'
WITH MOVE 'BookStore' TO 'C:\Apress\Recipes\Mirror\BookStore.mdf',
     MOVE 'BookStore_log' TO 'C:\Apress\Recipes\Mirror\BookStore_log.ldf',
     NORECOVERY
GO
```

706

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在这之后，回到主体数据库服务器，创建一个事务日志备份：

```
BACKUP LOG BookStore
TO DISK =
'C:\Apress\Recipes\Mirror\BookStore_tlog.trn'
WITH INIT
```

然后在将事务日志备份文件手动复制到第二个SQL Server实例中之后，将事务日志备份还原到数据库的镜像副本（再一次使用NORECOVERY选项）：

```
RESTORE LOG BookStore
FROM DISK = 'C:\Apress\Recipes\Mirror\BookStore_tlog.trn'
WITH FILE = 1, NORECOVERY
```

现在，有了处于NORECOVERY状态的另一个数据库副本，这样，就做好了示例场景的下一步的准备——创建数据库镜像会话。

25.3.3 创建数据库镜像会话

一旦还原了数据库并且使镜像服务器处于恢复模式，就可以使用ALTER DATABASE命令启动镜像会话。通过两个步骤来完成它（如果在这个场景中要使用见证SQL Server实例，就需要3个步骤）。首先，要在镜像SQL Server实例上执行ALTER DATABASE，将它设置为带有主体服务器端点的伙伴。之后，在主体SQL Server实例上使用ALTER DATABASE，设置镜像伙伴和见证端点的位置。

使用ALTER DATABASE启用数据库镜像的语法如下：

```
ALTER DATABASE database_name
[PARTNER { = 'partner_server'
 | FAILOVER
 | FORCE_SERVICE_ALLOW_DATA_LOSS
 | OFF
 | RESUME
 | SAFETY { FULL | OFF }
 | SUSPEND
| TIMEOUT integer
          } |
   WITNESS { = 'witness_server'
           | OFF
           }]
```

表25-2描述了这个命令的参数。这些选项中没有详细讲解的几个选项将在本章稍后展示。

表25-2 ALTER DATABASE参数

参 数	描 述
database_name	这个参数定义参与到镜像会话中的数据库名称(主体服务器和镜像服务器的名称必须相同)
partner_server	这个参数指定伙伴服务器的名称，要遵循下列格式：TCP://fully_qualified_domain _name:port
FAILOVER	FAILOVER选项手动进行从主体数据库到镜像数据库的故障转移。这个选项要求SAFETY选项为FULL
FORCE_SERVICE_ALLOW_DATA_LOSS	FORCE_SERVICE_ALLOW_DATA_LOSS允许不通过进行完全同步最新的事务，强制故障转移到镜像数据库（这就意味着可能会丢失数据）。这个操作要求主体服务器数据库为不可用状态，SAFETY选项为OFF，并且没有指定见证
OFF	OFF选项会停止数据库镜像会话

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（续）

参　数	描　述
RESUME	启动暂停的数据库镜像会话
SAFETY { FULL \| OFF }	SAFETY设置有两个值，FULL或OFF。当SAFETY为FULL时，数据库镜像会话以异步的模式进行，要求在主体数据库的事务允许提交之前要写入镜像数据库。当SAFETY为OFF时，镜像会话是同步的，意思是主体上的事务在提交之前，不会等待应用到镜像数据库上（这就意味着可能会丢失数据）
SUSPEND	SUSPEND模式暂停数据库镜像会话
TIMEOUT integer	TIMEOUT选项指定服务器实例在认为连接已经不可用（这将引发故障转移）前，要等待多久才能接收从其他伙伴返回的PING消息（伙伴服务器之间的心跳）。最小等待时间为5秒，默认值为10秒
witness_server	这是见证服务器的名称，要遵循下列格式：TCP://fully_qualified_domain_name:port
OFF	OFF将见证从数据库镜像会话中删除

继续这个示例的场景，在镜像SQL Server应例中，执行随后的命令，通过引用主SQL Server实例和TCP端口号（端点监听的地址）去启动镜像的进程：

```
-- 在镜像SQL Server实例上设置
-- 默认SAFETY为FULL——异步模式
ALTER DATABASE BookStore
    SET PARTNER = 'TCP://NODE2.JOEPROD.COM:5022'
GO
```

下一步，在主体SQL Server实例上执行ALTER DATABASE，指定镜像服务器的名称以及TCP端口号：

```
-- 在主体SQL Server实例上启用镜像会话
-- 默认SAFETY为FULL——异步模式
ALTER DATABASE BookStore
    SET PARTNER = 'TCP://NODE1.JOEPROD.COM:5022'
GO
```

708

在设置完镜像和主体服务器后，可以选择增加或不增加见证服务器，需要在主体SQL Server实例上进行配置，如这个示例所展示的：

```
-- 在主体SQL Server实例上启用见证
-- 默认SAFETY为FULL——异步模式
ALTER DATABASE BookStore
    SET WITNESS = 'TCP://NODE3.JOEPROD.COM:5022'
GO
```

在这个示例中，配置了拥有可选的见证服务器的镜像。任何主体数据库上的数据修改或架构改变都会记录到镜像数据库。镜像数据库在活动是不可用的，除非它通过自动或手动的故障转移变为主体数据库（我们将在本章稍后对其讨论和展示）。

解析

在这个示例中，通过使用ALTER DATABASE命令启动数据库镜像会话。在镜像SQL Server实例上启动。使用SET PARTNER执行ALTER DATABASE：

```
ALTER DATABASE BookStore
    SET PARTNER = 'TCP://NODE2.JOEPROD.COM:5022'
GO
```

这个命令中的PARTNER指定了主体数据库SQL Server实例的名称以及SQL Server实例的监听端点端口。而前面提到的端点配置是以伙伴（不是见证）身份监听端口5022。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下一步，在主体数据库SQL Server实例上，使用ALTER DATABASE命令设置数据库镜像伙伴，这时指定镜像数据库节点和要监听的TCP端口：

```
ALTER DATABASE BookStore
    SET PARTNER = 'TCP://NODE1.JOEPROD.COM:5022'
GO
```

包含了主体数据库的SQL Server实例也是需要配置数据库镜像会话的见证的地方。而在前面的技巧中，是在见证SQL Server实例上创建了数据库镜像端点。当使用ALTER DATABASE 和SET WITNESS时，就（从主体数据库SQL Server实例）指定了见证服务器的机器名称和要监听的TCP端口：

```
ALTER DATABASE BookStore
    SET WITNESS = 'TCP://NODE3.JOEPROD.COM:5022'
GO
```

现在，已经配置好数据库镜像会话了。BookStore数据库上添加或修改的任何数据库对象都将转移到镜像副本。如果主体数据库变为不可用，将会发生故障转移，镜像数据库角色将转为主体数据库。但是在开始这些任务之前，我将快速地总结一下前面几个技巧实现了些什么。

25.4 设置总结

前面的3个技巧是设置数据库镜像的一般步骤，在这里我们一步步地回顾一下如何完成它。 709

(1) 首先，在镜像SQL Server实例中，使用CREATE ENDPOINT创建了一个端点，并且指定角色为PARTNER，使用了TCP端口5022。

(2) 在主体SQL Server实例上，使用PARTNER角色和监听端口5022创建了端点（因为这些是独立的服务器，只要这个端口是可用的，就可以在它们上面使用相同的TCP端口）。

(3) 下一步，在见证SQL Server实例上，创建了角色为WITNESS且监听端口为5022的端点。

(4) 回到主体SQL Server实例上，使用ALTER DATABASE将BookStore数据库（要镜像的数据库）设为FULL恢复模式（如果它已经是FULL恢复模式，就不需要这个步骤了）。

(5) 仍然在主体SQL Server实例上，在BookStore数据库上执行完全数据库备份。

(6) 然后，在镜像SQL Server实例上，使用NORECOVERY选项还原了数据库，让它处于可以从镜像进程中接收事务的状态。如果在最后一次完全备份之后，在主体数据库上进行了其他的事务日志备份，那么这些事务日志备份也要附加到镜像的、还原的副本上。

(7) 在主体SQL Server实例上，在BookStore数据库上执行事务日志备份。

(8) 在镜像SQL Server实例上，使用NORECOVERY选项还原事务日志备份，让它处于可以从镜像进程中接收事务的状态。

(9) 仍然在镜像SQL Server实例上，执行ALTER DATABASE...SET PARTNER命令，指出完全限定的主体服务器名称和主体SQL Server实例端点监听的TCP端口。

(10) 在主体SQL Server实例上，执行ALTER DATABASE...SET PARTNER，指出完全限定的镜像服务器名称和镜像SQL Server实例端点监听的TCP端口。

(11) 最后，仍然从主体上执行ALTER DATABASE...SET WITNESS，指出完全限定的见证服务器名称和见证SQL Server实例端点监听的TCP端口。

在进行完这些之后，启动了数据库镜像会话，主体数据库上的修改将被记录到处于等待状态的镜像数据库。见证服务器会密切注意主体和镜像数据库之间的连接，确保如果发生任何问题，都会采取适当的动作（比如说自动故障转移）。在讨论故障转移之前，在下面的部分将讨论镜像会话中的各种

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

运行模式，以前如何对它们进行修改和控制。

25.5　运行数据库镜像

数据库镜像会话运行有3种模式：高可用性（前面的示例中使用的）、高安全性或高性能性。

高可用性模式，意思是在主体数据库上提交事务，在事务可以提交之前，要求主体和镜像数据库都是可用的。这个模式也要求有见证服务器，它允许自动地故障转移。主体数据库的拥有者通过仲裁
710 做出决定，至少要有两个可以互相通信的服务器。如果见证与镜像失去联系，但是与主体保持联系，主体数据库将仍然保持它的角色。如果见证与主体失去联系，但仍然可以看到镜像，在高可用性模式下镜像会假定为主体角色。如果不管任何原因见证处于不可用状态，主体和镜像将形成仲裁，并且保持它们之前的角色。简而言之——它们两者作为一个仲裁，并且仲裁决定哪一个伙伴控制主体数据库。

高安全性模式，就像高可用性模式，意思是在主体数据库上提交事务，在事务可以提交之前，要求主体和镜像数据库同时可用。但是不像高可用性模式，它不包含见证。这个意思是只可以进行手动的故障转移，不能进行自动的故障转移。高安全性模式仍然与镜像数据库一起形成仲裁（两个伙伴服务器一起），但是如果镜像数据库变为不可用，SQL Server会使数据库不可用（这意味着停止服务）。这是因为高安全性模式要求镜像按顺序提交事务。

上述的两种模式建议数据安全性和可用性作为工作的重点。但是，这个功能也带来了性能上的开销。如果你的镜像数据库存在数据可观的更新活动，所有主体数据库上的事务在事务可以提交之前，必须等待镜像副本的接受。

高性能模式，它允许在主体数据库上进行异步更新（在提交事务之前不等待镜像），没有见证服务器。这个模式强调了事务的速度，而不是数据的可用性（由于缺少手工或自动的故障转移）和最小的数据恢复能力（异步修改允许在镜像数据库上存在事务丢失）。

注解　在SQL Server 2008中，数据库镜像允许自动地数据流压缩和日志传送缓冲优化。

在下一个技巧中，将展示如何使用ALTER DATABASE配置高可用性、高安全性以及高性能模式。

25.5.1　改变运行模式

高可用性和高安全性模式都使用FULL安全模式（当启动镜像会话时，这是默认模式）。但是可以通过使用ALTER DATABASE...SET SAFETY将这个设置关闭。这个命令可以使用两个选项：OFF或FULL。

在第一个示例中，通过在主体SQL Server实例上执行这个命令，将指定的镜像会话的安全转换为OFF：

```
ALTER DATABASE BookStore SET SAFETY OFF
```

第二个示例展示了将安全转换为原来的状态，并且从高性能转换为高可用性模式：

```
ALTER DATABASE BookStore SET SAFETY FULL
```

高安全性模式没有在这里展示，因为它也是将安全模式启用为FULL的，只是在数据库镜像会话中没有使用见证。

解析

在这个示例中，通过引用数据库名称以及新安全模式（OFF或FULL），镜像会话安全被关闭然后又
711 被打开。如果使用了见证且安全为FULL，数据库镜像会话将运行在高可用性模式。如果没有使用见证，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

但安全仍为FULL，则数据库镜像会话运行在高安全性模式。如果安全为OFF，则数据库处于异步、高性能模式。请看表25-3中这些不同模式的总结。

表25-3 数据库镜像运行模式

模 式	安全配置	是否有见证
高可用性	FULL	有
高安全性	FULL	没有
高性能	OFF	没有

提示 作为最佳实践，在关键任务数据库中使用同步、高可用性模式。只在可以从其他机制或来源轻易恢复丢失的数据的数据库上使用异步、高性能模式。如果选择了同步模式，为了利用自动故障转移，尽可能使用见证服务器。

25.5.2 进行故障转移

故障转移涉及主体和镜像数据库角色的转换，镜像副本变为主体数据库并且主体数据库变为镜像数据库。在故障转移时既有的数据库连接断开，然后连接的应用程序必须连接到新的主体数据库上（通过.NET的功能，连接串可以是数据库镜像可识别的）。

可以通过使用ALTER DATABASE...SET PARTNER FAILOVER命令，在同步的高性能或高可用性模式下手动地将参与到镜像会话中的数据库进行故障转移。

在这个示例中，从主体服务器启动故障转移（将会在操作后变为镜像服务器）：

```
USE master
GO
ALTER DATABASE BookStore SET PARTNER FAILOVER
```

故障转移操作实际运行多长时间，依赖于镜像副本上记录的事务前滚所花费的时间。

如果数据库会话运行在异步、高性能模式下，就不可以手动进行故障转移。相反地，如果主体数据库变为不可用，你可以等待数据库再一次变为可用，或者强制将服务转移到镜像副本。使用ALTER DATABASE...SET PARTNER FORCE_SERVICE_ALLOW_DATA_LOSS命令强制服务。在强制服务之后，镜像数据库将前滚记录的事务（并且在异步模式下，主体数据库可能在停运之前在转变中丢失一些事务）。然后镜像数据库转变为主体数据库。

警告 只有在绝对必要的情况下才可以使用镜像数据库上的强制服务，因为在不可用的主体数据库上会丢失数据。

712

在这个示例中，使用了异步、高性能模式的数据库会话中的镜像数据库被强制服务（要求镜像会话中实际的主体数据库不可用）。

```
ALTER DATABASE ReportCentralDB SET PARTNER FORCE_SERVICE_ALLOW_DATA_LOSS
```

解析

这些示例展示了故障转移的选项，它依赖于数据库镜像会话的模式。你可以决定进行手动故障转移，例如，为了对主体数据库服务器上的活动进行维护。

但是对于异步、高性能模式，如果主体数据库变为不可用，则只有在绝对必要的情况下，才能寄希望于在镜像会话中使用强制服务。这是因为在不可用的主体数据库上会丢失未发送的事务。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

25.5.3　暂停或继续镜像会话

如果镜像主体数据库中有大量的数据更新，同步更新到镜像会成为影响数据库性能的瓶颈，你可以使用ALTER DATABASE...SET PARTNER SUSPEND临时暂停镜像会话。这个选项保持主体数据库为可用状态，并且将更新保存到日志中，一旦镜像会话继续，日志将会发送到镜像数据库。数据库镜像会话应该只暂停一小段时间，因为事务日志会持续增长，如果事务日志文件是固定容量的或是不停地增长直至充满磁盘（如果事务日志是非固定容量的），就会引发磁盘已满的问题。

在这个示例中，暂停主体服务器上的BookStore数据库镜像会话：

```
ALTER DATABASE BookStore SET PARTNER SUSPEND
```

然后，通过查询主体服务器上的系统目录视图sys.database_mirroring确认数据库的状态：

```
SELECT mirroring_state_desc
FROM sys.database_mirroring
WHERE database_id = DB_ID('BookStore')
```

这个查询返回：

```
mirroring_state_desc
SUSPENDED
```

下一个示例展示了继续数据库镜像会话，这会引发镜像数据库使用挂起的日志事务去进行同步操作：

```
ALTER DATABASE BookStore SET PARTNER RESUME
```

解析

可以不通过完全删除数据库镜像会话去暂停或继续镜像会话。使用这个技巧中的方法可以消除可能会发生在高活动率的数据库上的性能瓶颈。但是要留心事务日志的容量，并且不要禁用镜像会话的时长超过它实际需要的时间。

713

25.5.4　停止镜像会话和删除端点

在前面的示例中展示了暂时停止和继续镜像会话，如果想完会删除它，可以使用ALTER DATABASE...SET PARTNER OFF命令。

在这个示例中，停止并删除BookStore数据库镜像（删除镜像的元数据）：

```
ALTER DATABASE BookStore SET PARTNER OFF
```

可以使用DROP ENDPOINT命令删除所有SQL Server实例上的镜像端点，例如：

```
-- 在见证服务器上执行
DROP ENDPOINT JOEPROD_Witness
-- 在镜像服务器上执行
DROP ENDPOINT JOEPROD_Mirror
-- 在主体服务器上执行
DROP ENDPOINT JOEPROD_Mirror
```

解析

使用ALTER DATABASE...SET PARTNER OFF停止和删除数据库镜像会话。断开在主体数据库上的连接，但是在删除镜像会话后仍允许恢复常规活动的连接。这个操作将镜像副本保留为恢复状态，你可以恢复或删除它。如果你希望将这个镜像恢复原状，必须从头开始，一步步地在这个被破坏的镜像会话上重新设置主体、镜像和见证。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

25.6 镜像和配置选项

本章最后一组技巧将展示如何执行下列操作。

- 使用系统目录视图sys.database_mirroring监视数据库镜像的状态。
- 使用ALTER DATABASE...SET PARTNER TIMEOUT命令配置连接超时时限。

你将在后面的内容中学习如何监视数据库镜像会话的当前状态。

25.6.1 监视镜像状态

可以通过查询系统目录视图sys.database_mirroring确认镜像会话状态。

例如，在主体数据库上执行这个视图，来显示镜像的状态、当前数据库的角色、安全级别（将在下一技巧中描述）以及连接到主体数据库的见证数据库的状态：

714

```
SELECT mirroring_state_desc, mirroring_role_desc, mirroring_safety_level_desc,
mirroring_witness_state_desc
FROM sys.database_mirroring
WHERE database_id = DB_ID('BookStore')
```

这个查询返回：

mirroring_state_desc	mirroring_role_desc	mirroring_safety_level_desc	mirroring_witness_state_desc
SYNCHRONIZED	PRINCIPAL	FULL	CONNECTED

解析

当安全模式为默认的FULL时，SYNCHRONIZED状态意味着主体数据库和包含相同数据的镜像数据库。可以在视图中看到下面一些状态。

- SYNCHRONIZING，意思是主体正在发送镜像仍在附加过程中的日志记录。
- SUSPENDED，意思是数据库的镜像副本不可用，已经发生了错误，或是手动操作数据库到这个状态的。在SUSPENDED状态，主体数据库运行但没有发送日志记录到镜像数据库。
- PENDING_FAILOVER，意思是有手动故障转移的请求，但是还没有执行的时候。
- DISCONNECTED，意思是伙伴失去了与见证和其他伙伴的连接。

25.6.2 配置连接超时时限

数据库镜像使用默认的连接超时时限10秒。如果不能在10秒后建立一个连接，就意味着发生了故障，并且依赖于数据库的角色（主体、镜像或见证）或镜像会话模式（同步、异步），将会发生故障转移或镜像关闭。

如果数据库镜像会话中的网络延迟过早地引发故障，可以使用ALTER DATABASE...SET PARTNER TIMEOUT命令配置连接超时时限。这个命令以秒为单位配置超时时限（最小允许5秒）。

在这个示例中，在主体服务器上创建15秒的连接超时时限：

```
ALTER DATABASE BookStore SET PARTNER TIMEOUT 15
```

可以通过查询系统目录视图sys.database_mirroring确认新的设置：

```
SELECT mirroring_connection_timeout
FROM sys.database_mirroring
WHERE database_id = DB_ID('BookStore')
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

715
```
mirroring_connection_timeout
15
```

解析

在这个技巧中，使用ALTER DATABASE...SET PARTNER TIMEOUT命令修改连接超时时限：

```
ALTER DATABASE BookStore SET PARTNER TIMEOUT 15
```

当镜像会话处于活动状态时，会在参与的服务器之间发送PING通信消息。当一个服务器实例等待时间超过配置的超时时限时，就引发故障。这个故障的反应依赖于服务器的角色、仲裁是如何定义的（两个服务器仍能互相监视）以及数据库镜像模式（高可用性、高安全性或高性能）。

716

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 26 章

数据库快照 26

数据库快照是只读的、某一特定时间点的静态数据库副本。可以像连接到其他数据库一样连接到这些快照，使用它们可以生成报表、做测试、做培训或恢复数据。在处理巨大的或有潜在危险的数据库更新之前，当可能需要撤销你的工作时，可以使用数据库快照作为预防措施。

本章将展示如何创建、查询以及删除数据库快照，还有如何使用数据库快照进行数据恢复。

26.1 快照基础

可以从用户数据库创建数据库快照，提供在快照生成的特定时间点的数据的只读视图。可以为一个数据库创建多个快照，例如，可以在每天、每周或每个月结束时为数据库创建快照。

数据库快照也是节省空间的，因为它们使用了“稀疏文件”（sparse file）。稀疏文件在最初创建时是不包含用户数据的。为了维护源快照数据，快照保留了最少的磁盘空间。最初创建快照时，数据库快照不会在源数据库中产生所有数据额外的副本，但经过一段时间后源数据库发生数据改变，这时变化前的数据将保存到稀疏文件中。然后快照将包含就像快照刚创建时一样的数据库内容。

在快照上的查询将从快照或数据库或它们两者中返回数据。未更新的源数据库的数据仍从源数据库中被检索。但是，在创建快照数据库之后，源数据库中发生变化的数据将从快照中被检索。

当源数据库中变化的数据百分比接近100%时，数据库快照也将接近最初创建快照时原始数据库的大小。请记住，如果源数据库上相同的数据被修改多次，则数据库快照上没有额外的更新。一旦源数据库上的数据页被更新，则变化前的数据页只移动一次到快照数据库。

当决定是否使用快照时，请记住快照的一些局限性。例如，不能为系统数据库创建快照。并且数据库快照不能像常规数据库一样进行备份、还原、附加或分离操作。同时快照也给源数据库增加性能开销。这是因为你将看到每个引起数据页移动到快照文件的修改增加了I/O活动。如果同一个源数据库上有多个快照，需要页面更新的所有快照都会增加I/O活动。

除了这些局限，数据库快照提供了一些很好的方法，用来保留时点数据、从源数据库分离出报表查询、允许快速数据恢复。

随后一些技巧将实际地展示数据库快照操作。

26.1.1 创建和查询数据库快照

使用CREATE DATABASE命令创建数据库快照。

这个命令的语法如下：

```
CREATE DATABASE database_snapshot_name
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    ON
        (NAME = logical_file_name,
        FILENAME = 'os_file_name' ) [ ,...n ]
    AS SNAPSHOT OF source_database_name
```

表26-1描述了这个命令的参数。

表26-1 CREATE DATABASE...AS SNAPSHOT参数

参 数	描 述
database_snapshot_name	这个参数指定要创建的数据库快照名称
(NAME = logical_file_name,FILENAME = 'os_file_name') [,...n]	logical_file_name是源数据库数据文件的逻辑文件名。os_file_name是要创建的快照文件的物理文件名。对于所有源数据库数据文件，必须为其定义一个快照文件
source_database_name	这个参数定义快照基于的源数据库

在这个技巧的示例中，为数据库AdventureWorks生成快照：

```
CREATE DATABASE AdventureWorks_Snapshot_Oct_08_2008
ON
(  NAME = AdventureWorks2008_Data,
   FILENAME =
'C:\Apress\Recipes\AdventureWorks_Snapshot_Oct_08_2008.mdf')
AS SNAPSHOT OF AdventureWorks

GO
```

下一步，在数据库AdventureWorks中执行更新操作，来展示数据库快照的功能：

```
USE AdventureWorks
GO

UPDATE HumanResources.Department
SET GroupName = 'Materials'
WHERE Name='Production'
GO
```

然后，查询AdventureWorks源数据库中的表HumanResources.Department来确认所作的更改：

```
SELECT GroupName
FROM AdventureWorks.HumanResources.Department
WHERE Name='Production'
```

718

这个查询返回：

```
GroupName
Materials
```

接着，查询快照：

```
SELECT GroupName
FROM AdventureWorks_Snapshot_Oct_08_2008.HumanResources.Department
WHERE Name='Production'
```

快照返回GroupName在更改之前的、快照创建时的原始数据：

```
GroupName
Manufacturing
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个技巧中，使用CREATE DATABASE命令创建数据库快照：

```
CREATE DATABASE AdventureWorks_Snapshot_Oct_08_2008
```

ON子句包括数据库AdventureWorks的数据文件的逻辑名，随后是新数据库快照数据文件（由于快照是只读的，所以不需要事务日志文件）的物理路径和文件名：

```
ON
(  NAME = AdventureWorks_Data,
  FILENAME = 'C:\Apress\Recipes\AdventureWorks_Snapshot_Oct_08_2008.mdf')
```

AS子句指定了快照基于哪一个数据库：

```
AS SNAPSHOT OF AdventureWorks
```

创建了快照之后，在数据库AdventureWorks上执行了更新操作。在这种情况下，SQL Server将变化前的数据页复制到数据库快照文件。

在创建了快照之后，对请求源数据库中发生变化的数据的快照的查询将从快照数据库上读取。这个“写入时复制”的功能允许快照文件容量保持相对较小，意思是只有那些受影响变化的数据才需要保存到快照数据文件中。

26.1.2 删除数据库快照

使用DROP DATABASE命令删除数据库快照。

语法如下：

```
DROP DATABASE database_snapshot_name
```

这个命令只使用了一个参数：数据库快照的名称。

下一个示例展示了删除前面技巧中创建的数据库快照：

```
DROP DATABASE AdventureWorks_Snapshot_Oct_08_2008
```

719

解析

在这个技巧中通过使用DROP DATABASE删除了快照。这个操作从SQL Server实例中删除了快照，也删除了相关的物理快照文件。

26.1.3 从数据库快照恢复数据

考虑一下这个不寻常的场景：你接到了数据库最终用户的电话，他告诉你他不小心更新了表中所有行的一列值。他修改的数据库非常大，从数据库备份还原数据首先需要从磁带上读出备份文件。准备好备份文件后，必须把数据库还原到另外的数据库名上，然后使用INSERT...SELECT将丢失的数据插入到生产数据库中。经受了操作的所有痛苦之后，你可能还发现没有足够的磁盘空间同时保存备份文件和增加的数据库还原副本。

现在想象你已经为数据库在执行巨大的数据更新之前创建了周期性的快照。由于源数据库中数据的易变性，数据库快照可能只消耗了完整数据库还原需要空间的一部分。使用快照，可以通过将快照数据库中的数据更新到源数据库，来还原/更新受前面示例影响的数据。

或者，如果你可以承受的话，也可以覆盖既有的源数据库，即通过使用RESTORE...FROM DATABASE_SNAPSHOT命令，恢复数据到最后一次的快照。使用RESTORE...FROM DATABASE_SNAPSHOT，SQL Server将用数据库快照覆盖既有的源数据库。RESTORE...FROM DATABASE_SNAPSHOT命令只与快照一起使用。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注解 如需使用RESTORE命令的其他功能，请看第29章。

从数据库快照恢复的语法如下：

```
RESTORE DATABASE <database_name>
FROM DATABASE_SNAPSHOT = <database_snapshot_name>
```

这个命令有两个参数：希望还原到的源数据库的名称，以及希望从哪里还原的数据库快照的名称。

使用RESTORE...FROM DATABASE_SNAPSHOT，将丢失在最后一次快照之后源数据库中所有的数据修改，只能恢复到创建快照时间时的数据。但是，在不得不只更新在创建快照之后发生修改的信息时，与常规的数据库还原操作相比，RESTORE...FROM DATABASE_SNAPSHOT操作能够显著地节省时间。可以达到与常规数据库还原相似的结果，数据库状态被恢复到创建快照时的内容。

警告 只有在可以承受丢失最后一次快照之后源数据库中发生的所有修改的情况下，才可以恢复到快照！这个方法对于“临时”数据库非常有用，比如说用作培训或测试的数据库。虽然数据库快照对恢复数据来说很便捷，但是数据库快照也不应该作为更好的数据恢复计划的替代品。

720 如果计划通过使用数据库快照在SQL Server实例中恢复数据，注意其他的数据库快照（不从那里恢复的快照）必须在RESTORE...FROM DATABASE_SNAPSHOT操作之前删除。否则你将看到类似下面这样的出错信息：

```
消息3137，级别16，状态4，第2行
无法恢复数据库。可能是因为错误地指定了主名称或快照名称，没有删除所有其他快照，或者缺少文件。
消息3013，级别16，状态1，第2行
RESTORE DATABASE正在异常终止。
```

除了数据库快照，数据库中的全文本目录也必须在数据库快照RESTORE...FROM DATABASE_SNAPSHOT操作之前删除，并且你的源数据库不能包含只读或离线的文件组。

注解 由于从快照文件还原会破坏事务日志备份序列（请看第29章），因此在执行RESTORE操作之后进行一个完全数据库备份是一个好主意。

在下一个技巧中，首先创建新的数据库，然后再将AdventureWorks数据库中的数据插入到表中：

```
IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'TSQL_AW')
BEGIN
   CREATE DATABASE TSQL_AW
END
GO

USE TSQL_AW
GO

SELECT BusinessEntityID, CreditCardID, ModifiedDate
INTO dbo.PersonCreditCard
FROM AdventureWorks.Sales.PersonCreditCard
```

然后，将在TSQL_AW数据库中新建数据库快照：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE DATABASE TSQL_AW_Oct_09_2008
ON
(  NAME = TSQL_AW,
  FILENAME = 'C:\Apress\Recipes\TSQL_AW_Oct_09_2008.mdf')
AS SNAPSHOT OF TSQL_AW
GO
```

下一步，删除dbo.PersonCreditCard表中所有的行（没有使用WHERE子句）：

```
USE TSQL_AW
GO
DELETE dbo.PersonCreditCard
```

这个查询返回：

```
(19118行受影响)
```

721

执行查询验证TSQL_AW数据库中发生的变化：

```
SELECT COUNT(*)
FROM dbo.PersonCreditCard
```

这个查询返回：

```
0
```

下一步，将数据库TSQL_AW恢复到数据库快照时的状态。要记住，在源数据库进行RESTORE操作时，不允许有会话连接到源数据库上，所以这个示例将数据库上下文改变为数据库master。数据库TSQL_AW在操作期间是离线的：

```
USE master
GO

RESTORE DATABASE TSQL_AW
FROM DATABASE_SNAPSHOT = 'TSQL_AW_Oct_09_2008'
```

再一次执行验证查询，来查看数据是否已恢复：

```
USE TSQL_AW
GO

SELECT COUNT(*)
FROM dbo.PersonCreditCard
```

这个查询返回：

```
19118
```

26

解析

RESTORE...FROM DATABASE_SNAPSHOT命令允许撤销在创建指定的数据库快照之后对源数据库所做的所有修改。这个操作也比常规还原操作的执行时间少（更多关于此主题的内容，请看第29章）。

在这个技巧中，表中所有的行都被意外地删除了。然后使用了数据库快照将数据恢复到创建数据库快照时的状态。在执行RESTORE...FROM DATABASE_SNAPSHOT操作过程中，数据库是离线的，并不可用。在操作完成后，所有在数据库快照之后对数据库TSQL_AW所做的修改都将丢失。

722

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第27章 链接服务器和分布式查询

链接服务器为SQL Server提供了从远程数据源访问数据的能力。使用链接服务器，可以执行查询、数据修改以及远程过程调用。远程数据源可以是同类（意思是数据源是另一个SQL Server实例）或异类的（来自其他关系型数据库产品和数据源，比如DB2、Access、Oracle、Excel和文本文件）。跨平台联结或检索数据的查询是跨平台查询。使用跨平台查询，可以访问遗留数据库系统，而不需要对既有的数据源进行合并和迁移。

通过OLE DB访问接口连接远程的数据源。OLE DB由微软开发，是用来提供到各种不同的数据源的一致性访问的一组组件对象模型COM（component object model）接口。为了建立从SQL Server实例到另一数据源的访问，需要选择适当的OLE DB访问接口。OLE DB访问接口的设计方式决定在分布式查询中实现何种分布式查询操作。

因此概括地说，链接服务器是建立到远程数据源的连接的一种途径。依赖用来设置链接服务器的OLE DB驱动，可以执行分布式查询来检索数据，或在远程数据源上执行操作。

分布式查询也可以不通过定义链接服务器来运行，例如，通过使用T-SQL函数OPENROWSET。除了不通过链接服务器查询远程数据源，OPENROWSET允许BULK读取ASCII、Unicode以及二进制文件。使用OPENROWSET和BULK，可以从文本文件中读取表状数据，或使用它将ASCII、Unicode或二进制类型文件导入一个大数据类型列及一行中，比如varchar(max)、nvarchar(max)或varbinary(max)。

SQL Server还提供了SYNONYM对象，使用它可以通过较短的名称来引用长名称的对象。通常在使用长标识符时有用，尤其在引用了4部分组成的链接服务器名称的分布式查询中，使用较短的名称来引用数据源特别有用。

本章包括这些技巧：创建链接服务器、执行分布式查询、使用OPENROWSET和BULK从文本文件中读取数据以及使用新的SYNONYM对象。

27.1 链接服务器基础

下面一组技巧将展示如何使用链接服务器。具体将展示如何完成下列操作。

- 创建链接服务器连接到另一个SQL Server实例。
- 配置链接服务器的属性。
- 查看SQL Server实例上配置的链接服务器的信息。
- 删除链接服务器。

首先，讨论如何使用系统存储过程sp_addlinkedserver来创建新链接服务器。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

27.1.1 为另一 SQL Server 实例创建链接服务器

链接服务器允许从SQL Server实例中查询外部数据源。外部数据源可以是不同的SQL Server实例，也可以是非SQL Server数据源，比如说Oracle、MS Access、DB2或MS Excel。

使用系统存储过程sp_addlinkedserver来创建链接服务器。语法如下：

```
sp_addlinkedserver
     [ @server= ] 'server' [ ,
     [ @srvproduct= ] 'product_name' ]
     [ , [ @provider= ] 'provider_name' ]
     [ , [ @datasrc= ] 'data_source' ]
     [ , [ @location= ] 'location' ]
     [ , [ @provstr= ] 'provider_string' ]
     [ , [ @catalog= ] 'catalog' ]
```

表27-1描述了这个系统存储过程的参数。

表27-1 sp_addlinkedserver参数

参 数	描 述
server	这是链接服务器的本地名称。也允许使用实例名称，例如MYSERVER\SQL1
product_name	这是OLE DB数据源的产品名。对于SQL Server实例来说，product_name是'SQL Server'
provider_name	这是OLE DB访问接口的唯一可编程标识。当没有指定它时，访问接口名称是SQL Server数据源。SQL Server显式的provider_name是SQLNCLI（Microsoft SQL Native Client OLE DB Provider）。Oracle的是MSDAORA，Oracle 8或更高版本的是OraOLEDB.Oracle。MS Access和MS Excel的是Microsoft.Jet.OLEDB.4.0。IBM DB2的是DB2OLEDB，以及ODBC数据源的是MSDASQL
data_source	这是特定OLE DB访问接口解释的数据源。对于SQL Server，这是SQL Server（servername或servername\instancename）的网络名称。对于Oracle，这是SQL*Net别名。对于MS Access和MS Excel，这是文件的完整路径和名称。对于ODBC数据源，这是系统DSN名称
location	这是由特定OLE DB访问接口解释的位置
provider_string	这是OLE DB 访问接口特定的连接字符串。对于ODBC连接，这是ODBC连接字符串。对于MS Excel，这是Excel 5.0
catalog	catalog的定义变化基于OLE DB访问接口的实现。对于SQL Server，这是可选的数据库名称，对于DB2，这个目录是数据库的名称

在多SQL Server实例的网络环境中，链接服务器提供了方便的方式共享SQL Server数据，而不是必须要通过物理地推送或拉回数据和复制架构的方式来实现共享。

724

提示 在本章中，我讲解了在SQL Server实例之间通信的示例。对于异类的数据源，比如DB2、Access以及Oracle，参数会有很多变化。如果需要更全面了解sp_addlinkedserver选项，请看SQL Server联机丛书中的主题“sp_addlinkedserver（Transact-SQL）”。

连接到异类数据源使用的配置会基于OLE DB访问接口而发生变化。如果只是连接到不同的SQL Server实例，微软把它变得很简单。在这个示例中，我展示了如何创建连接到另一SQL Server实例的链接服务器：

```
EXEC sp_addlinkedserver
          @server= 'JOEPROD',
          @srvproduct= 'SQL Server'
```

也可以创建连接到SQL Server命名实例的链接服务器，例如：

```
EXEC sp_addlinkedserver
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
@server= 'JOEPROD\NODE2',
@srvproduct= 'SQL Server'
```

解析

添加连接到外部数据源的链接服务器允许执行分布式查询（分布式查询将在27.3节中介绍）。当添加SQL Server链接服务器到一个SQL Server实例时，不管它是默认的还是命名的实例，微软都把它变得很简单，只需要server和product_name值。

关于连接到SQL Server实例的安全性方法，当创建新的链接服务器时，使用当前用户的登录名安全凭证（SQL或Windows）来连接到链接服务器。也可以为链接服务器创建显示的远程登录名映射，稍后会讨论该内容。

27.1.2　配置链接服务器属性

在创建链接服务器之后有许多不同的设置可以用来配置它。表27-2描述了这些设置。

表27-2　链接服务器属性

设　置	描　述
collation compatible	如果确认SQL Server实例与远程SQL Server拥有相同的排序规则就启用这个设置。由于SQL Server不再需要对数据源之间的字符列执行比较操作，把它们假定为相同的排序规则，这样做可以提升性能
collation name	如果启用了use remote collation并且是非SQL Server的数据源，则collation name指定远程服务器排序规则的名称。这个排序规则名称必须是SQL Server所支持的
connect timeout	这个设置指定在超时发生之前到链接服务器的连接会尝试多少秒。如果数值为0，sp_ configure的remote query timeout的服务器值用来当作默认值
data access	如果这个选项设置为启用，就允许分布式查询访问
lazy schema validation	如果这个选项设为true，则架构不会在查询开始时去检测远程表。尽管这样会减少远程查询的开销，但是如果架构发生了变化并且你没有检测架构，比如说查询中引用的对象不能与查询命令进行通信，就会生成错误
query timeout	这个设置指定查询等待的超时值（秒数）。如果这个值为0，则query wait选项使用sp_configure值
rpc	这个设置启用来自服务器的远程过程调用
rpc out	这个设置启用到服务器的远程过程调用
remote proc transaction promotion	启用这个选项（默认是启用的），可通过MS DTC管理调用远程存储过程启动分布式事务
use remote collation	这个设置指定是使用远程服务器排序规则(true)还是本地服务器排序规则(false)

使用系统存储过程sp_serveroption来修改链接服务器属性。语法如下：

```
sp_serveroption [@server =] 'server'
     ,[@optname =] 'option_name'
   ,[@optvalue =] 'option_value'
```

表27-3描述了这个系统存储过程的参数。

表27-3　sp_serveroption参数

参　数	描　述
server	要配置属性的链接服务器的名称
option_name	要配置的选项
option_value	选项的新值

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在这个技巧中，链接服务器JOEPROD\NODE2的查询超时设置将修改为60秒：

```
EXEC sp_serveroption
   @server = 'JOEPROD\NODE2' ,
   @optname = 'query timeout',
   @optvalue = 60
```

726

解析

在这个技巧中，链接服务器JOEPROD\NODE2的查询超时限制被修改为60秒。名为server的第一个参数指定了链接服务器的名称。第二个参数option_name指定了要配置的选项，第三个参数option_value配置了新值。

27.1.3 查看链接服务器信息

可以使用系统目录视图sys.servers查看SQL Server实例中定义的链接服务器。例如：

```
SELECT name, query_timeout, lazy_schema_validation
FROM sys.servers
WHERE is_linked = 1
```

这个查询返回：

```
name              query_timeout     lazy_schema_validation
JOEPROD\NODE2     60                1
```

解析

系统目录视图sys.servers可以用来检索关于SQL Server实例中定义的链接服务器信息。可以从sys.servers查看到的其他选项包括：product、provider、data_source、location、provider_string、catalog、is_linked、is_remote_login_enabled、is_rpc_out_enabled、is_data_access_enabled、is_collation_compatible、use_remote_collation和collation_name。is_linked列在查询中限定只返回链接服务器（不包括本地SQL Server实例设置）。

27.1.4 删除链接服务器

系统存储过程sp_dropserver用来删除链接服务器。sp_dropserver的语法如下：

```
sp_dropserver [ @server= ] 'server'
     [ , [ @droplogins= ] { 'droplogins' | NULL} ]
```

表27-4描述了这个系统存储过程的参数。

表27-4 sp_dropserver的参数

参 数	描 述
server	这个参数定义从SQL Server实例中删除的链接服务器名称
droplogins	如果指定droplogins，则在删除链接服务器之前要删除登录名映射（本章稍后介绍）

27

这个技巧展示了删除链接服务器操作：

```
EXEC sp_dropserver
   @server= 'JOEPROD',
@droplogins= 'droplogins'
```

727

解析

这个技巧展示了使用系统存储过程sp_dropserver从SQL Server实例中删除链接服务器。在第二个

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

参数中指定的droplogins选项，将在删除链接服务器之前删除既有的登录名映射（下面几个技巧将介绍链接服务器登录）。如果在删除登录之前尝试删除链接服务器，将得到如下消息：

```
仍有对服务器的远程登录.
```

27.2　链接服务器登录名

在下面的3个技巧中，将展示如何使用链接服务器登录名映射。具体而言，将讲解如何完成下列操作。

- 创建链接服务器登录名映射。
- 查看配置在SQL Server实例中的链接服务器登录名映射。
- 删除链接服务器登录名映射。

首先，讨论如何使用系统存储过程sp_addlinkedsrvlogin来创建登录名映射。

27.2.1　添加链接服务器登录名映射

当在链接服务器上执行分布式查询时，SQL Server会将本地登录和凭据映射到链接服务器。基于远程数据源的安全性，凭据可能会被接受或是拒绝。当执行sp_addlinkedserver并且创建了链接服务器后，默认的行为是使用本地登录凭据（SQL或Windows）去访问链接服务器上的数据。即便没有足够的权限连接到链接服务器，在尝试执行分布式查询之前，SQL Server不会去检测链接服务器的安全性。由于SQL Server实例中安全性配置、登录和数据库用户的多样性，你可能需要从本地登录到不同的远程登录设置不同的映射。

登录名映射信息存储在定义链接服务器的SQL Server实例中，使用系统存储过程sp_addlinkedsrvlogin来创建登录名映射。语法如下：

```
sp_addlinkedsrvlogin [ @rmtsrvname = ] 'rmtsrvname'
    [ , [ @useself = ] 'TRUE' | 'FALSE' | 'NULL']
    [ , [ @locallogin = ] 'locallogin' ]
    [ , [ @rmtuser = ] 'rmtuser' ]
    [ , [ @rmtpassword = ] 'rmtpassword' ]
```

表27-5描述了这个系统存储过程的参数。

表27-5　sp_addlinkedsrvlogin的参数

参　数	描　述
rmtsrvname	这个参数定义要添加登录名映射的本地链接服务器
useself	当使用true值时，使用本地SQL或Windows登录名连接到远程服务器名。如果设为false，存储过程sp_addlinkedsrvlogin的locallogin、rmtuser和rmtpassword参数将应用到新的映射中
locallogin	这是映射到远程登录名的SQL Server登录或Windows用户的名称。如果这个参数置为NULL，映射将应用SQL Server实例中的所有本地登录名
rmtuser	这个参数是用来连接到链接服务器的用户／登录名的名称
rmtpassword	这个参数是用来连接到链接服务器的用户／登录名的密码

在这个技巧中，在链接服务器JOEPROD\NODE2中为所有本地用户创建了登录名映射——映射到登录名test：

```
EXEC sp_addlinkedsrvlogin
     @rmtsrvname = 'JOEPROD\NODE2',
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    @useself = 'false' ,
    @locallogin = NULL, -- 应用到所有本地登录名
    @rmtuser = 'test',
    @rmtpassword = 'test1!'
```

解析

在这个技巧中，使用系统存储过程sp_addlinkedsrvlogin显式地创建登录名映射。第一个参数@rmtsrvname包含了要连接到的链接服务器的名称。第二个参数@useself置为false值，因此将使用在远程服务器的@rmtuser和@rmtpassword参数中定义的登录名和密码。@locallogin置为NULL，意思是所有本地SQL Server连接的登录名都会映射到test登录名上。现在，当在TESTSRV链接服务器上执行查询时，这些查询将以test远程用户身份运行。

27.2.2 查看链接登录名

可以查询系统目录视图sys.server_principals、sys.linked_logins和sys.servers来查看到远程登录的显式的本地登录名映射。正如下面这个查询所展示的：

```
SELECT s.name LinkedServerName, ll.remote_name, p.name LocalLoginName
FROM sys.linked_logins ll
INNER JOIN sys.servers s ON
   s.server_id = ll.server_id
LEFT OUTER JOIN sys.server_principals p ON
   p.principal_id = ll.local_principal_id
WHERE s.is_linked = 1
```

这个查询返回：

```
LinkedServerName        remote_name      LocalLoginName
JOEPROD\NODE2           test             NULL
```

解析

这个技巧通过查询系统目录视图sys.linked_logins、sys.servers及sys.server_principals检索到远程登录的显式的登录名映射。查询返回链接服务器的名称、远程数据源的远程登录名以及映射到它的本地登录名。在这个例子中，结果返回远程登录名test及本地登录名NULL（意思是所有本地连接将映射到远程test登录名）。

729

27.2.3 删除链接服务器登录名映射

使用系统存储过程sp_droplinkedsrvlogin删除链接服务器登录名映射。sp_droplinkedsrvlogin的语法如下：

```
sp_droplinkedsrvlogin [ @rmtsrvname= ] 'rmtsrvname' ,
   [ @locallogin= ] 'locallogin'
```

表27-6描述了这个系统存储过程的参数。

表27-6 sp_droplinkedsrvlogin参数

参 数	描 述
rmtsrvname	登录名映射的链接服务器名称
locallogin	从链接服务器删除的SQL Server登录名或Windows用户映射的名称

这个技巧展示了删除在前面技巧中创建的登录名映射：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
EXEC sp_droplinkedsrvlogin
   @rmtsrvname= 'JOEPROD\NODE2' ,
   @locallogin= NULL
```

解析

在这个技巧中，通过把链接服务器名称传送到第一个参数，把NULL值传送到第二个@locallogin参数，删除了所有本地用户的默认登录名映射。

27.3　执行分布式查询

本章进行到现在为止，展示了如何创建和配置链接服务器。在下面一组技巧中，将学习如何在连接到远程数据源的链接服务器上执行分布式查询。你被限定使用链接服务器来连接到远程数据源，并且下面的一些技巧也会展示如何使用命令（比如OPENQUERY和OPENROWSET）来访问外部数据。你也会学习如何为链接服务器名称创建和使用别名。

27.3.1　在链接服务器上执行分布式查询

分布式查询引用一个或多个链接服务器进行查询，在远程表、视图或存储过程上执行读取和修改操作。链接服务器支持的查询类型（SELECT、INSERT、UPDATE、DELETE、EXEC）取决于OLE DB访问接口提供的事务支持级别。可以通过在FROM子句中使用4部分组成的远程对象名称来引用链接服务器，或使用T-SQL命令OPENQUERY（OPENQUERY将在本章稍后介绍）来运行分布式查询。

730

使用4部分组成的名称引用链接服务器的基本语法如下：

```
linked_server_name.catalog.schema.object_name
```

表27-7描述了4部分组成的名称。

表27-7　链接服务器4部分组成的名称

部　分	描　述
linked_server_name	链接服务器的名称
catalog	目录（数据库）名称
schema	数据源对象的架构容器
object_name	数据库对象（例如视图、表、数据源或存储过程）

分布式查询从链接服务器上的动态管理视图sys.dm_os_performance_counters中选取了性能计数器值：

```
SELECT object_name, counter_name, instance_name,
cntr_value, cntr_type
FROM JOEPROD.master.sys.dm_os_performance_counters
WHERE counter_name = 'Active Transactions' AND
  instance_name = '_Total'
```

这个查询返回：

```
object_name          counter_name          instance_name   cntr_value      cntr_type
SQLServer:Databases  Active Transactions   Total           0               65792
```

下面的查询展示了在链接服务器（SQL Server命名实例）上执行系统存储过程。链接服务器是一个命名实例，因此完整的名称就放入方括号中：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
EXEC [JOEPROD\NODE2].master.dbo.sp_monitor
```

这个查询返回各种关于远程SQL Server实例的统计信息和结果集。

```
last_run                current_run             seconds
2005-09-02 22:47:26.770 2005-10-09 10:52:27.007 3153901

cpu_busy                io_busy                 idle
53(53)-0%               22(21)-0%               10433(10192)-0%

packets_received        packets_sent            packet_errors
182(154)                377(349)                0(0)

total_read              total_write             total_errors    connections
693(693)                201(201)                0(0)            8091(8079)
```

解析

可以看出，执行分布式查询只是使用4部分组成的名称来引用数据库对象。如果需要引用是SQL Server命名实例的链接服务器，就要把链接服务器名称放入方括号内。

731

27.3.2 创建和使用引用 4 部分组成的链接服务器名的别名

可以为数据库对象创建别名（包括存储过程、函数、表和视图）。可以在代码中引用这个别名，允许将一个长名称缩短或将底层的对象源的修改隐藏起来（例如，开发环境与生产环境的链接服务器名称的切换）。

通过使用CREATE SYNONYM来实现这个功能，它使用如下的简略语法：

```
CREATE SYNONYM [ schema_name. ] synonym_name
FOR < object >
```

表27-8描述了这个命令的参数。

表27-8 CREATE SYNONYM

参 数	描 述
[schema_name.] synonym_name	可选的架构名称及必需的同义词名称
object	要赋予别名的对象。可以为以下格式：server_name.database_name.schema_name.object_name 、 database_name.schema_name.object_name 或 schema_name.object_name

使用DROP SYNONYM命令删除同义词。语法如下：

```
DROP SYNONYM [ schema. ] synonym_name
```

这个命令需要一个可选的同义词架构和必需的同义词名称。

在这个技巧中，为链接服务器创建同义词：

```
CREATE SYNONYM dbo.PerfInfo
FOR JOEPROD.master.sys.dm_os_performance_counters
```

下一步，在查询的FROM子句中使用新同义词名称来引用链接服务器的同义词：

```
SELECT cntr_value
FROM dbo.PerfInfo
WHERE counter_name = 'Active Transactions' AND
instance_name = '_Total'
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在此之后，从数据库中删除同义词：

```
DROP SYNONYM dbo.PerfInfo
```

最后，用与前面相同的名称创建新的同义词，而这时指向不同的SQL Server实例：

```
CREATE SYNONYM dbo.PerfInfo
FOR [JOEPROD\NODE2].master.sys.dm_os_performance_counters
```

解析

在这个技巧中，创建了一个名为PerfInfo的同义词，用来表示4部分组成的链接服务器上的表名称。同义词通过允许使用更短的名称表示链接服务器4部分组成的名称，可以减少输入。然后为了查询底层的链接服务器中的表，在FROM子句中使用同义词PerfInfo。在这之后，删除了同义词（尽管在实际情况下你可能会保留这个同义词以备后用）。最后，用之前的名称创建了新的同义词，它引用了新的数据源。这意味着原来对dbo.PerfInfo的查询现在访问不同的SQL Server实例。同义词给了你不改变引用的同义词名称，而去修改底层数据源的能力。

732

27.3.3　使用 OPENQUERY 执行分布式查询

SQL Server为执行分布式查询提供除使用4部分组成的名称以外的其他方式。

OPENQUERY是在查询的FROM子句中像表一样被引用且对既有的链接服务器传递查询的函数。语法如下：

```
OPENQUERY ( linked_server ,'query' )
```

表27-9描述了这个命令的参数。

表27-9　OPENQUERY参数

参　数	描　述
linked_server_name	要查询的链接服务器名称
catalog	对链接服务器连接发出的实际查询

OPENQUERY命令通过发送以传递查询形式，而不是引用4部分组成的名称的链接服务器，来查询链接服务器。传递查询在远程服务器上完整地执行，并且将结果返回到调用的查询。

为什么使用这个方式而不是另一个？用于创建链接服务器的一些OLE DB访问接口，可能拥有多种在FROM子句中使用4部分组成的名称被引用的能力。OPENQUERY是检索分布式数据且4部分组成的名称的查询可能不正常工作时的另一种方式。使用OPENQUERY命令还可以避免潜在的跨服务器连接性能问题，允许查询完全在远程服务器中执行。

这个示例展示了使用与前面技巧中相同的查询来查询链接服务器，只是这次在OPENQUERY命令的第二个参数中，实际的查询在FROM子句使用了3部分的名称，而不是4部分：

```
SELECT cntr_value
FROM OPENQUERY ( [JOEPROD] ,
'SELECT object_name, counter_name, instance_name, cntr_value, cntr_type
FROM master.sys.dm_os_performance_counters
WHERE counter_name = ''Active Transactions'' AND
instance_name = ''_Total''' )
```

解析

在这个技巧中，OPENQUERY命令中的第一个参数是链接服务器的名称。第二个参数是查询本身。注意，在WHERE子句中包含了两个引号内的值，它的作用就是单引号。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

27.3.4 使用 OPENROWSET 执行临时的查询

就像OPENQUERY，在FROM子句中引用OPENROWSET命令，并且像SELECT语句中的表一样操作它。不像OPENQUERY的地方是，OPENROWSET会创建一个到数据源的临时的连接。它没有使用既有的链接服务器连接查询远程数据源。如果不希望为SQL Server实例的远程数据源保留链接服务器，这是一个有用的功能。

OPENROWSET的语法如下：

```
OPENROWSET
( { 'provider_name' ,
{ 'datasource' ; 'user_id' ; 'password' | 'provider_string' }
, { [ catalog. ] [ schema. ] object | 'query' }
```

表27-10描述了这个命令的参数。

表27-10 OPENROWSET参数

参 数	描 述
provider_name	OLE DB访问接口的唯一可编程标识符
datasource ; user_id ; password	OLE DB访问接口需要的连接字符串。可以指定datasource、user_id和password，或者指定访问接口字符串
provider_string	
catalog.schema.object \| query	返回结果的对象名称或要执行的查询

在这个技巧中，对SQL Server命名实例执行查询：

```
SELECT *
FROM OPENROWSET
('SQLNCLI','TESTSRV\NODE2';'test';'test1!',
'SELECT * FROM AdventureWorks.HumanResources.Department
WHERE GroupName = ''Research and Development''')
```

这个查询返回：

DepartmentID	Name	GroupName	ModifiedDate
1	Engineering	Research and Development	1998-06-01 00:00:00.000
2	Tool Design	Research and Development	1998-06-01 00:00:00.000
6	Research and Development	Research and Development	1998-06-01 00:00:00.000
18	Misc	Research and Development	2005-09-20 19:20:25.570

解析

在这个技巧中，使用了OPENROWSET去查询远程数据源，而不需要定义链接服务器。命令的第一个参数指定为SQLNCLI，这是Microsoft SQL Native Client OLE DB Provider的访问接口名称。第二个参数包括3个分号分隔的值——SQL Server实例名称、登录名及密码。命令的最后一个参数包含对远程SQL Server实例中AdventureWorks数据库的查询。

提示 远程访问需要启用sp_configure的Ad Hoc Distributed Queries选项，并将供应程序（provider）的Disallow Adhoc Access注册表选项设置为0。

OPENROWSET可以用在SELECT的FROM子句中，也可以用作INSERT、UPDATE或DELETE操作的目标表——取决于OLE DB访问接口是否支持更新。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

27.3.5　使用OPENROWSET BULK选项从文件中读取数据

正如第2章中的技巧所演示的，可以在OPENROWSET命令中使用新BULK选项从ASCII、Unicode或二进制文件查询数据。有了这个功能，可以查询文件以及使用数据修改语句的结果集——所有这些都不需要先从文件将数据物理地导入到SQL Server的表中。

在OPENROWSET中使用BULK选项的语法如下：

```
OPENROWSET
( { BULK 'data_file' ,
      { FORMATFILE = 'format_file_path' [ <bulk_options> ]
      | SINGLE_BLOB | SINGLE_CLOB | SINGLE_NCLOB }
} )
```

表27-11描述这个命令的参数。

表27-11　OPENROWSET...BULK的参数

参　数	描　述
data_file	这个参数定义要读取的文件名称和路径
format_file_path	这个参数指定格式化文件的名称和路径——它列出数据文件中的列定义。可以选择两种格式化文件格式——XML或非XML
bulk_options	这些选项定义如何读取数据和检索哪些行。细节内容请看表27-12
SINGLE_BLOB \| SINGLE_CLOB \| SINGLE_NCLOB	当指定这个选项时，将忽略格式化文件参数。作为替代，以单行单列值导入数据文件。例如，如果希望将文档或图像文件导入大数据类型列，则要指定其中一个标志。导入到varbinary(max)类型要指定SINGLE_BLOB对象，ASCII数据导入到varchar(max)数据类型指定SINGLE_CLOB，Unicode数据导入到nvarchar(max)类型要指定SINGLE_NCLOB

BULK选项语法如下：

```
<bulk_options> ::=
   [ , CODEPAGE = { 'ACP' | 'OEM' | 'RAW' | 'code_page' }]
   [ , ERRORFILE = 'file_name' ]
   [ , FIRSTROW = first_row ]
   [ , LASTROW = last_row ]
   [ , MAXERRORS = maximum_errors ]
   [ , ROWS_PER_BATCH = rows_per_batch ]
```

表27-12描述了这些选项。

735

表27-12　BULK选项

参　数	描　述
'ACP' \| 'OEM' \| 'RAW' \| 'code_page'	这个参数指定选择何种字符数据的源数据代码页转换为目标SQL Server代码页。OEM是默认值。ACP是ISO 1252代码页，RAW表示没有转换，code_page是指定的代码页编号
file_name	这个参数定义用来保存BULK进程拒绝的行的错误文件名称
first_row	这个参数指定结果集中载入的第一行（默认为1）。如果第一行包括列名称，你可以指定first_row为2跳过第一行
last_row	这个参数定义在结果集中载入的最后一行（结果集的最后一行，默认为0）
maximum_errors	这个参数提供在载入失败之前载入操作中最大的错误数量（默认为10）
rows_per_batch	这个值表示每个批处理导入的行数，虽然OPENROWSET总是以单个的批处理导入数据。指定这个值可以帮助查询处理器分配适当的资源。不过在大多数情况下，这个选项可以忽略或是设为0

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在这个技巧中，将展示两个从外部文本文件读取数据的示例。

第一个示例展示了使用SELECT语句从文本文件读取数据。文本文件中数据用逗号分隔，文件名为ContactType.txt：

```
21,Sales Phone Rep,2005-06-01 00:00:00
20,Sales Phone Manager,2005-06-01 00:00:00
```

这个文件中的列将使用名为ContactTypeFormat.Fmt的格式化文件来定义，这个格式化文件包含如下的格式化文件定义（SQL Server允许XML格式和常规文本格式文件）：

```
10.0
3
1 SQLCHAR 0 2 "," 1 ContactTypeID ""
2 SQLCHAR 0 20 "," 2 Name SQL_Latin1_General_CP1_CI_AS
3 SQLCHAR 0 19 "\r\n" 3 ModifiedDate ""
```

这个查询从SELECT查询的ContactType.txt文件中读取数据：

```
SELECT ContactTypeID, Name, ModifiedDate
FROM OPENROWSET( BULK 'C:\Apress\Recipes\ContactType.txt',
            FORMATFILE = 'C:\Apress\Recipes\ContactTypeFormat.Fmt',
            FIRSTROW = 1,
            MAXERRORS = 5,
            ERRORFILE = 'C:\Apress\Recipes\ImportErrors.txt' )
            AS ContactType
```

这个查询返回：

```
ContactTypeID      Name                 ModifiedDate
21                 Sales Phone Rep      2005-06-01 00:00:00
20                 Sales Phone Manager  2005-06-01 00:00:00
```

736

技巧中的第二个示例将把ContactType.txt文件导入到单列单行中（而不是像前面的查询那样把它分散到表状数据集中）。首先，创建表保存导入的文档：

```
-- 创建表保存导入的文档
CREATE TABLE dbo.ImportRepository
   (ImportHistoryID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
    ImportFile varchar(max) NOT NULL)
GO
```

下一步，使用OPENROWSET...BULK将数据导入到新行中：

```
INSERT dbo.ImportRepository
(ImportFile)
SELECT BulkColumn
FROM OPENROWSET( BULK 'C:\Apress\Recipes\ContactType.txt',
            SINGLE_CLOB) as ContactTypeFile
```

现在确认一下内容：

```
SELECT ImportFile
FROM dbo.ImportRepository
```

这个查询返回：

```
ImportFile
21,Sales Phone Rep,2005-06-01 00:00:00  20,Sales Phone Manager,2005-06-01 00:00:00
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

在这个技巧的第一个示例中，使用OPENROWSET BULK选项查询数据文件。SELECT子句包含了数据文件中的列，这些列是通过格式化文件定义的：

```
SELECT ContactTypeID, Name, ModifiedDate
```

然后将OPENROWSET命令包含在FROM子句中。BULK选项是这个命令中的第一个参数，随后是数据和数据格式化文件：

```
FROM OPENROWSET( BULK 'C:\Apress\Recipes\ContactType.txt',
FORMATFILE = 'C:\Apress\Recipes\ContactTypeFormat.Fmt',
```

另外还包括了3个选项，指定要导入的数据文件的第一行：

```
FIRSTROW = 1,
```

也指定了在导入失败之前允许发生错误的数量，以及保存被拒绝行的错误文件：

```
MAXERRORS = 5,
ERRORFILE = 'C:\Apress\Recipes\ImportErrors.txt' )
```

在右括号之后，为了在SELECT中使用，还需要一个表名称的别名：

```
AS ContactType
```

第二个示例使用OPENROWSET将单个文件的全部内容插入到单列单行中。在创建保存结果的表之后，
737 使用了INSERT SELECT：

```
INSERT dbo.ImportRepository
(ImportFile)
```

SELECT引用了系统列名称BulkColumn，当使用SINGLE_选项（BLOB、CLOB、NCLOB）时从OPENROWSET返回它：

```
SELECT BulkColumn
```

OPENROWSET放置在SELECT语句的FROM子句中，随后是文件的名称和SINGLE_CLOB选项（以ASCII文本的格式导入数据）：

```
FROM OPENROWSET( BULK 'C:\Apress\Recipes\ContactType.txt',
          SINGLE_CLOB) as ContactTypeFile
```

使用T-SQL导入文件（ASCII、Unicode或二进制）比在SQL Server 2005以前的版本中要容易得多。
738 对表执行查询，并且原始文件格式的结果显示到单列单行中（以逗号分隔的完整形式）。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第28章 查询性能调优

SQL Server查询性能调优需要多重途径。有一些关键因素会影响SQL Server查询性能，它包括下面所列举的内容。

- 数据库设计。这或许是影响查询性能和数据完整性最重要的因素了，设计决策影响读取和修改的性能。标准设计包括规范化OLTP数据库，它关注数据完整性、去除冗余以及建立多个实体之间的关系。这是一个适合快速事务处理的设计。你通常会在规范化OLTP设计中看到较多的表，这意味着在查询中要联结较多的表。另一方面，数据仓库设计经常使用非规范化的星形结构或雪花状结构设计。这些设计使用了中心事实表，它联结了两个或更多的描述维度表。对于雪花状设计，维度表也可以存在于关联到它的关系表上。这种设计的重点在于查询执行的速度而不是事务的快速更新。
- 适当的索引。表索引应当基于高优先级或频繁执行的查询。如果一个查询一天执行数千次且每次2秒内完成，则可以通过适当的索引使每次运行时间少于1秒，增加这个索引可以显著减少SQL Server实例上的I/O压力。应该根据需要创建索引，并且要删除不用的索引（本章会告诉你如何做）。如同大多数的改变一样，这需要一个权衡。表上的每一个索引都会给数据修改操作增加开销，而且如果SQL Server判定使用低效的索引甚至还会减慢SELECT查询的速度。在你最初设计数据库的时候，索引最好比较少些（至少要有聚集索引和外键的非聚集索引）。当有了在数据库中执行实际查询的更好的主意时才增加索引。索引需求是有机的，特别是在易变的、频繁更新的数据库中，因此应该灵活、迭代地增加和移除索引。
- 索引碎片。由于长期的数据修改，索引将会产生碎片。由于碎片增多，数据将会扩散到更多的数据页。查询需要检索的数据页越多，I/O需求就越高，查询就越慢。
- 配置。这个类别包括数据库、SQL实例和操作系统配置。不适当的配置（像启用自动收缩和自动关闭数据库）会影响应用程序的性能。
- 最新的统计信息。正如第22章讲到的，AUTO_CREATE_STATISTICS数据库选项激活了SQL Server自动产生关于列中值的分布的统计信息。如果关闭这个选项，统计信息就会过期。由于SQL Server依赖统计信息去决定如何执行查询，如果SQL Server基于过期的统计信息进行判定，它可能就会选择一个非最佳的计划。
- 硬件。我曾经花费了一整天的时间试图使执行3秒的查询降到1秒。不论我怎样加索引，或去查询我做的修改，都没办法让时间减少。原因是结果集中的行太多，而限制因素则是I/O。过了几个月，我将该数据库迁移到更高配置的生产服务器上。执行查询的时间就一直低于1秒了。这说明了好的硬件确实很重要。对处理器体系结构、可用内存和磁盘子系统的选择会显著影响查询性能。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 网络吞吐量。获得查询结果所花的时间会受到缓慢的或者不稳定的网络连接的影响。这不意味着只要查询执行的慢，你就去责怪可怜的网络工程师——但是你需要把这个潜在的影响因素记录在你要研究的范围列表中。

在本章中，我将展示用于对查询性能进行评估和故障排除的命令及方法。你也将学习到如何定位存在碎片的索引和过期的统计信息，以及评估数据库中索引的使用情况。

提示　由于这是一本关注于T-SQL的书，我不会讲述像SQL Server Profiler、图形执行计划、系统监视器以及数据库引擎优化顾问这些为性能调优提供支持的图形化接口工具。这些都是非常有用的工具——因此我仍然建议你将它们作为整个性能调优策略的一个组成部分，也是对本章中学习的T-SQL命令和方法的补充。

对于SQL Server 2008带来的新功能，本章将演示下面所列举的内容。

- 使用资源调配器来控制工作负载和关联的CPU和内存资源。
- 使用新过滤统计信息的改进功能为数据子集创建统计信息。
- 显示几乎相同的查询（除了非参数化文本值外都相同的查询）或相同查询执行计划的聚合查询统计信息。
- 使用sp_create_plan_guide_from_handle系统存储过程基于在查询计划缓存中既有的查询计划创建计划指南。

本章还将讲解几个不同种类的查询性能主题，包括：如何使用sp_executesql来替代动态SQL，如何不修改查询本身对查询应用查询提示，以及如何强制查询使用指定的查询执行计划。

28.1　查询性能技巧

在开始讨论可以用来评估查询性能的命令和工具之前，首先简要地介绍一些基本查询性能调优指导方针。查询性能是个很大的主题，而且在许多章节中我努力在各种内容讲解中都包含一些小的技巧。
740 既然本章讨论的查询性能独立于特定的对象，下面将详细描述构建SQL Server查询的查询性能最佳实践（注意，索引技巧会在本章稍后介绍）。

- 在SELECT查询中，只返回需要的列。查询中存在的列越少，则使用的I/O和网络带宽就越少。
- 在减少使用列的同时，也应该考虑一下减少行。使用WHERE子句帮助减少查询返回的行。不要让应用程序在只需要显示前10行时却返回20 000行。
- 保证FROM子句在你的控制之下。JOIN到单个查询中的每一张表都会增加额外的开销。我没办法给出明确数量的注意事项，因为它依赖于你数据库的设计、容量大小和用于联结查询的列。但是，经过多年的经验，我发现大量的查询功能虽是正确的，但是执行的时间太长。尽管使用单个查询去执行复杂的操作显得很方便，但不要低估短小的查询的能力。如果存储过程中有一个执行很长时间的非常大的查询，我通常会尝试将这个查询分解为几个更小的中间结果集。这通常会显著地加快最后结果集的生成。
- 只在需要有序的结果时使用ORDER BY。大结果集的排序操作会导致额外的开销。如果它在你的查询中不是必需的，就不使用它。
- 避免在FROM、WHERE和HAVING子句中隐式数据类型的转换。当谓词中底层的数据类型不匹配就会发生隐式数据类型转换，并且是由SQL Server自动转换的。例如，Java应用程序发送Unicode文本到非Unicode列。对于应用程序每秒处理成百个事务，而这些隐式转换确实会增加处理时间。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 如果唯一行不是必需的，就不要使用DISTINCT或UNION（而是使用UNION ALL）。
- 小心在真空环境下的测试。当在测试SQL Server实例中开发数据库时，将有代表性的数据集填充到表中是非常重要的。这个意思是你应该将与实际生产环境数量相当的行填充到表中，也包含有代表性的数据值集。不要在你的开发数据库内使用假数据，然后指望查询的性能会与生产环境的相似。SQL Server性能非常依赖索引和统计信息，并且SQL Server将基于包含在表内实际的值做出决策。如果测试数据不是有代表性的较真实的数据，那么当生产环境中的查询效果与在测试数据库上看到的查询效果不同时，你会感到非常惊奇。
- 当在游标与基本集合的方法之间进行选择时，最好选择后者。如果你必须使用游标，一定要尽快关闭和释放它们。
- 在更复杂的数据库驱动的应用程序中，一些情况下使用查询提示是有必要的。然而，当底层的数据量或分布情况发生变化时，它们经常会延长有效性。所以，要谨慎地使用提示来避免覆盖SQL Server的决策处理。
- 应避免使用嵌套视图。我常常会看创建的视图同时引用了其他视图，而这个所引用的对象在调用的视图中已经被引用过了！由于查询的复杂性，这个重复和重叠经常会造成未优化的查询计划。
- 我将这点放入了第10章，我也觉得值得在这里重复它。存储过程经常产生比常规即席查询调用更好的性能。存储过程也提升了查询执行的稳定性（重用既有的查询执行计划）。如果你有一个执行时间不理想的查询，考虑一下将这个查询封装到存储过程里。

741

当阅读关于SQL Server性能调优方面的内容时（正如你现在做的），小心“从不”和“总是”这两个词。当阅读查询调优方面的内容时，对“它依赖……”这个回答不要觉得奇怪，结果可能变得多种多样。做实验时（当然，是在测试环境中）思维开阔并放轻松。经常提出问题并且不要思维定式。

28.2 捕捉和评估查询性能

在下面的一组技巧中，将展示如何捕捉和评估查询性能。还将演示几个其他用来返回查询执行计划细节信息的T-SQL命令。

28.2.1 使用 sys.dm_exec_requests 捕捉执行的查询

除了在SQL Server Profiler中捕捉查询，也可以通过查询动态管理视图sys.dm_exec_requests来捕捉当前正在执行的查询的SQL，正如这个技巧所展示的：

```
SELECT r.session_id, r.status, r.start_time, r.command, s.text
FROM sys.dm_exec_requests r
CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) s
WHERE r.status = 'running'
```

这个查询会捕捉到所有当前正在执行的查询，即使这个查询是用来捕捉其他查询的：

session_id	status	start_time	command	text
55	running	2008-10-16 13:53:52.670	SELECT	SELECT r.session_id, r.status, r.start_time, r.command, s.text FROM sys.dm_exec_requests r CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) s WHERE r.status = 'running'

28

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解析

动态管理视图sys.dm_exec_requests返回关于SQL Server实例中正在执行的请求的信息。

查询的第一行选取了会话ID、查询的状态、开始时间、命令类型（例如SELECT、INSERT、UPDATE、DELETE）以及实际的SQL文本：

```
SELECT r.session_id, r.status, r.start_time, r.command, s.text
```

在FROM子句中，动态管理视图sys.dm_exec_requests与动态管理函数sys.dm_exec_sql_text交叉应用。该函数使用动态管理视图sys.dm_exec_requests中的sql_handle，并返回关联的SQL文本：

```
FROM sys.dm_exec_requests r
CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) s
```

然后WHERE子句指定返回当前正在运行的进程：

```
WHERE r.status = 'running'
```

742

28.2.2　使用 T-SQL 命令查看评估的查询执行计划

了解SQL Server如何执行查询有助于你决定如何才能修复一个执行性能差的查询。可以通过查看查询的执行计划（图形化或基于命令的）标识一些普通的操作，它包括如下内容：

- 批处理中最高消耗的查询和最高消耗的运算符；
- 索引或表扫描（访问堆或索引的所有页面）与使用查找（只访问选定的行）相比较；
- 丢失的统计信息或其他警告；
- 高消耗排序或计算活动；
- 查找操作使用非聚集索引访问行，但需要访问聚集索引来检索没有被非聚集索引覆盖的列；
- 从运算符传递大量行计数到另一运算符；
- 隐式数据类型转换（在Convert元素的Implicit属性等于1的XML计划中标识出来）。

在SQL Server中，有3个命令可以用来查看关于SQL语句或批处理的查询执行计划的详细信息：SET SHOWPLAN_ALL、SHOWPLAN_TEXT和SET SHOWPLAN_XML。这些命令的输出有助于你了解SQL Server如何计划去处理和执行查询，标识使用的表联结类型及访问的索引等信息。例如，使用这些命令的输出，可以查看SQL Server在查询中是否使用了指定的索引，如果是这样，它是否通过索引查找（使用非聚集索引来检索操作中选择的行）或索引扫描（操作中检索的所有索引行）检索数据。

SET SHOWPLAN_ALL、SET SHOWPLAN_TEXT、SET SHOWPLAN_XML命令启用之后可以不通过执行查询提供计划信息，使用它可以在真正执行查询之前调整相关表的查询或索引。

这些命令通过不同的方式返回信息。SET SHOWPLAN_ALL以包括多列和多行的列表格式返回评估的查询计划。输出包含每个操作中估计的I/O或CPU、操作的行数、操作开销（与它自身及查询的变化相关）以及使用的物理和逻辑运算符等信息。

注解　逻辑运算符描述了SQL Server必须在查询执行中执行的概念上的操作。物理运算符用来执行逻辑运算符实际的操作。例如查询中的逻辑操作INNER JOIN在实际的查询执行中会被转换成嵌套循环联结的物理操作。

SET SHOWPLAN_TEXT命令在单列中返回数据，每个操作占多行。也可以使用SET SHOWPLAN_XML命令以XML的格式返回查询执行计划。

这些命令的语法非常相似，这些命令当设为ON时就启用，设为OFF就禁用：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SET SHOWPLAN_ALL  { ON | OFF }
SET SHOWPLAN_TEXT { ON | OFF}
SET SHOWPLAN_XML  { ON | OFF }
```

743

这个技巧的示例展示在AdventureWorks数据库中通过使用SET SHOWPLAN_TEXT，然后使用SET SHOWPLAN_XML，来返回评估的查询执行计划：

```
SET SHOWPLAN_TEXT ON
GO

SELECT p.Name, p.ProductNumber, r.ReviewerName
FROM Production.Product p
INNER JOIN Production.ProductReview r ON
   p.ProductID = r.ProductID
WHERE r.Rating > 2
GO

SET SHOWPLAN_TEXT OFF
GO
```

它返回下面评估的查询执行计划结果：

```
StmtText

SELECT p.Name, p.ProductNumber, r.ReviewerName
FROM Production.Product p
INNER JOIN Production.ProductReview r ON
   p.ProductID = r.ProductID
WHERE r.Rating > 2

(1 行受影响)

StmtText
  |--Nested Loops(Inner Join, OUTER REFERENCES:([r].[ProductID]))
       |--Clustered Index Scan (OBJECT:(
          [AdventureWorks].[Production].[ProductReview].
          [PK_ProductReview_ProductReviewID] AS [r]),
          WHERE:([AdventureWorks].[Production].[ProductReview].[Rating]
          as [r].[Rating]>(2)))
       |--Clustered Index Seek
          (OBJECT:([AdventureWorks].[Production].[Product].
          [PK_Product_ProductID] AS [p]),
          SEEK:([p].[ProductID]=[AdventureWorks].[Production].
          [ProductReview].[ProductID] as [r].[ProductID]) ORDERED FORWARD)

(3 行受影响)
```

下一个示例以XML的格式返回评估的查询执行计划结果：

```
SET SHOWPLAN_XML ON
GO

SELECT p.Name, p.ProductNumber, r.ReviewerName
FROM Production.Product p
INNER JOIN Production.ProductReview r ON
   p.ProductID = r.ProductID
WHERE r.Rating > 2
GO

SET SHOWPLAN_XML OFF
GO
```

744

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

它返回如下的内容（实际的输出有一页多，这里只是部分结果）：

```
<ShowPlanXML xmlns="http://schemas.microsoft.com/sqlserver/2004/07/showplan"
Version="1.1" Build="10.0.1424.2">
  <BatchSequence>
    <Batch>
      <Statements>
...
 <RelOp NodeId="0" PhysicalOp="Nested Loops" LogicalOp="Inner Join" EstimateRows="3"
  EstimateIO="0" EstimateCPU="1.254e-005" AvgRowSize="105"
  EstimatedTotalSubtreeCost="0.00996111" Parallel="0" EstimateRebinds="0"
  EstimateRewinds="0">
              <OutputList>
               <ColumnReference Database="[AdventureWorks]" Schema="[Production]"
                Table="[Product]" Alias="[p]" Column="Name" />
              <ColumnReference Database="[AdventureWorks]" Schema="[Production]"
               Table="[Product]" Alias="[p]" Column="ProductNumber" />
             <ColumnReference Database="[AdventureWorks]" Schema="[Production]"
              Table="[ProductReview]" Alias="[r]" Column="ReviewerName" />
           </OutputList>
...
```

解析

可以使用SHOWPLAN_ALL、SHOWPLAN_TEXT或SHOWPLAN_XML来调整T-SQL查询和批处理。这些命令不通过实际地执行查询显示估计的执行计划。可以使用这些命令返回的信息采取动作，来改进查询性能。例如，为在搜索或联结条件中使用的列增加索引。观察一下输出，可以发现SQL Server是否使用了希望的索引，以及如果是这样，SQL Server是否使用了索引查找、索引扫描或表扫描操作。

在这个技巧中，SET SHOWPLAN_TEXT和SET SHOWPLAN_XML设置为ON，然后跟随GO：

```
SET SHOWPLAN_TEXT ON
GO
```

然后计算引用了Production.Product和Production.ProductReview的查询。这两张表通过在列ProductID上使用INNER联结进行了联结，只返回生产率大于等于2的产品：

```
SELECT p.Name, p.ProductNumber, r.ReviewerName
FROM Production.Product p
INNER JOIN Production.ProductReview r ON
   p.ProductID = r.ProductID
WHERE r.Rating > 2
```

在查询的结尾SHOWPLAN设置为OFF，这样这个连接上的后续查询就不会继续执行SHOWPLAN了。

观察输出的代码片段，你可以看到执行INNTER JOIN（逻辑操作）使用的嵌套循环联结（物理操作）：

745

```
|--Nested Loops(Inner Join, OUTER REFERENCES:([r].[ProductID]))
```

还可以从这个输出看到，通过使用PK_ProductReview_ProductReviewID主键聚集索引执行聚集索引扫描，来从表ProductReview中返回数据：

```
|--Clustered Index Scan (OBJECT:([AdventureWorks].[Production].[ProductReview].
[PK_ProductReview_ProductReviewID] AS [r]),
```

然后，通过使用聚集索引查找从表Product检索数据：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
|--Clustered Index Seek(OBJECT:([AdventureWorks].[Production].[Product].
[PK_Product_ProductID] AS [p]),
```

SET SHOWPLAN_XML命令以XML文档格式返回估计的查询计划，显示与SHOWPLAN_TEXT相似的数据。通过使用属性和元素格式化XML数据。

例如，元素RelOp的属性显示Nested Loops的物理操作和Inner Join的逻辑操作，以及其他统计信息（如估计的操作影响的行）：

```
<RelOp NodeId="0" PhysicalOp="Nested Loops" LogicalOp="Inner Join" EstimateRows="3"
EstimateIO="0" EstimateCPU="1.254e-005" AvgRowSize="105"
EstimatedTotalSubtreeCost="0.00996111" Parallel="0" EstimateRebinds="0"
EstimateRewinds="0">
```

XML文档使用了一个指定的架构定义格式，它定义返回的XML元素、属性和数据类型。这个架构可以通过下面的URL来查看：http://schemas.microsoft.com/sqlserver/2004/07/showplan/showplanxml.xsd。

28.2.3 查看执行运行时信息

SQL Server提供了4个用来返回查询和批处理执行的统计信息和其他信息的命令：SET STATISTICS IO、SET STATISTICS TIME、SET STATISTICS PROFILE和SET STATISTICS XML。

不像SHOWPLAN命令，STATISTICS命令返回在SQL Server中实际执行的查询信息。SET STATISTICS IO命令用来返回执行的语句产生的磁盘活动（也就是I/O）。SET STATISTICS TIME命令返回分析、编译和执行批处理中的所有语句所花费的时间，单位是毫秒。

SET STATISTICS PROFILE和SET STATISTICS XML与SET SHOWPLAN_ALL和SET SHOWPLAN_XML功能相同，但是只返回实际的（不是估计的）执行计划信息和实际的查询结果。

这些命令的语法是相似的，ON启用统计，OFF关闭统计：

```
SET STATISTICS IO { ON | OFF }

SET STATISTICS TIME { ON | OFF }

SET STATISTICS PROFILE { ON | OFF }

SET STATISTICS XML { ON | OFF }
```

746

在第一个示例中，首先启用STATISTICS IO，然后执行从表Sales.SalesOrderHeader和表Sales.SalesTerritory中通过地域联结累加应付款的总额的查询：

```
SET STATISTICS IO ON
GO

SELECT  t.name TerritoryNM,
      SUM(TotalDue) TotalDue
FROM Sales.SalesOrderHeader h
INNER JOIN Sales.SalesTerritory t ON
   h.TerritoryID = t.TerritoryID
WHERE OrderDate BETWEEN '1/1/2003' AND '12/31/2003'
GROUP BY t.name
ORDER BY t.name

SET STATISTICS IO OFF
GO
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回下面的部分结果：

```
TerritoryNM        TotalDue
Australia          4547123.2777
Canada             8186021.9178
...
Southwest          11523237.5187
United Kingdom     4365879.4375

表'Worktable'。扫描计数1，逻辑读取39次，物理读取0次，预读0次，lob逻辑读取0次，lob物理读取0次，lob预读0次。

表'SalesOrderHeader'。扫描计数1，逻辑读取703次，物理读取0次，预读0次，lob逻辑读取0次，lob物理读取0次，lob预读0次。

表'SalesTerritory'。扫描计数1，逻辑读取2次，物理读取1次，预读0次，lob逻辑读取0次，lob物理读取0次，lob预读0次。
```

对于相同的查询，用SET STATISTICS TIME取代SET STATISTICS IO将返回下面的（部分）结果：

```
Territory          NM TotalDue
Australia          4547123.2777
...
Southeast          3261402.9982
Southwest          11523237.5187
United Kingdom     4365879.4375

SQL Server分析和编译时间：
   CPU时间 = 20毫秒，占用时间 = 21毫秒。

(10行受影响)

SQL Server执行时间：
   CPU时间 = 30毫秒，占用时间 = 24毫秒。
```

747

解析

SET STATISTICS命令返回实际的查询或批处理执行的信息。在这个技巧中，SET STATISTICS IO返回查询引用的表有关的逻辑的、物理的和大对象读取事件的信息。对于存在性能问题的查询（基于业务需求和“问题”的定义），可以使用SET STATISTICS IO查看哪里I/O操作频繁。例如，在这个技巧的结果集中，可以看到SalesOrderHeader存在更高数量的逻辑读取：

```
...
表'SalesOrderHeader'。扫描计数1，逻辑读取686次，物理读取0次，预读0次，lob逻辑读取0次，lob物理读取0次，lob预读0次。
...
```

注意高物理的（从磁盘中读取）或逻辑的读取值（从数据缓存中读取）——即使物理读取值是零而逻辑的是一个很高的值。也要寻找工作表（已经在这个技巧中看过）：

```
表'Worktable'。扫描计数1，逻辑读取39次，物理读取0次，预读0次，lob逻辑读取0次，lob物理读取0次，lob预读0次。
```

通常会在使用了GROUP BY、ORDER BY、散列联结和UNION操作的查询中看到工作表。工作表在查询的执行期创建在tempdb中，并且在SQL Server完成操作时自动删除。

在这个技巧的第二个示例中，使用SET STATISTICS TIME来显示查询的分析和编译时间（在实际的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

查询结果之前显示），以及实际的执行时间（在查询结果之后显示）。这个命令在计算查询从头到尾消耗的时间总和时是有用的，允许查看预编译是否花费比预计的更长的时间，或是在实际的查询执行时是否发生延迟。

其他两个STATISTICS命令，SET STATISTICS PROFILE和SET STATISTICS XML，返回与SET SHOWPLAN_ALL和SET SHOWPLAN_XML相似的信息，只不过结果基于实际的执行计划，而不是估计的执行计划。

28.2.4 查看性能统计信息和缓存的查询计划

在这个技巧中，展示了使用SQL Server动态管理视图和函数查看性能统计信息和缓存的查询计划。

提示 SQL Server 2008为收集和分析性能统计信息引入各种各样的改进功能。例如，数据收集器使用存储过程、SQL Server集成服务和SQL Server代理作业，来收集数据并将它加载到管理数据仓库。

在这个示例中，是一个简单的查询，它返回AdventureWorks数据库上表Sales.SalesPerson中所有的行。在执行它之前，要清空过程缓存以便可以在这个示范中更容易地识别查询（请记住，应该只在测试SQL Server实例上清空过程缓存）：

748

```
DBCC FREEPROCCACHE
GO

SELECT BusinessEntityID, TerritoryID, SalesQuota
FROM Sales.SalesPerson
```

现在，将查询动态管理视图sys.dm_exec_query_stats，它包含关于缓存在SQL Server实例中查询的统计信息。这个视图包含了sql_handle，将它作为动态管理函数sys.dm_exec_sql_text的一个输入。这个函数用来返回T-SQL语句的文本：

```
SELECT   t.text,
       st.total_logical_reads,
       st.total_physical_reads,
       st.total_elapsed_time/1000000 Total_Time_Secs,
       st.total_logical_writes
FROM sys.dm_exec_query_stats st
CROSS APPLY sys.dm_exec_sql_text(st.sql_handle) t
```

这个查询返回如下信息：

text	total_logical_reads	total_physical_reads	Total_Time_Secs	total_logical_writes
SELECT BusinessEntityID...	2	0	0	0

解析

这个技巧展示了清空过程缓存，然后执行查询，几秒后执行完成了。完成之后，查询动态管理视图sys.sm_exec_query_stats来返回缓存执行计划的统计信息。

SELECT子句检索查询中T-SQL文件上的信息：逻辑读取、物理读取、总的占用时间（秒）以及逻辑写入（如果有）：

```
SELECT   t.text,
       st.total_logical_reads,
       st.total_physical_reads,
       st.total_elapsed_time/1000000 Total_Time_Secs,
       st.total_logical_writes
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

总的占用时间列以微秒为单位，因此为了返回完整的秒数要把它除以1 000 000。

在FROM子句中，为了检索缓存查询的SQL文本，动态管理视图sys.dm_exec_query_stats与动态管理函数sys.dm_exec_sql_text进行了交叉应用：

```
FROM sys.dm_exec_query_stats st
CROSS APPLY sys.dm_exec_sql_text(st.sql_handle) t
```

这个信息对于识别高强读取或高强写入的查询是有用的，帮助确定哪个查询是需要调优的。要牢记，这个技巧中的查询只能检索仍然在缓存中的查询信息。这个查询返回总和，但是sys.dm_exec_query_stats也包含跟踪读取和写入最大、最小和最后的值的列。注意，sys.dm_exec_query _stats还有其他有用的列，它们能表示CPU时间（total_worker_time、last_worker_time、min_worker_time以及max_worker_time）和.NET CLR对象执行时间（total_clr_time、last_clr_time、min_clr_time、max_clr_time）。

749

28.2.5　基于查询或计划模式查看聚合的性能统计信息

前面的技巧演示了使用sys.dm_exec_query_stats动态管理视图来查看查询性能。只要查询计划保留在缓存中，这个动态管理视图中的统计信息就可以显示。对于使用存储过程或预定义计划的应用程序，sys.dm_exec_query_stats可以为整个聚合的统计信息和资源利用提供准确的描述。然而，如果应用程序发送未预定义的查询文本和没有适当参数化的文本值，每个几乎相同查询的变种将生成独立的统计信息，使统计信息在相关性和聚合方面有所不同。

例如，假设应用程序发送如下3个独立的SELECT语句：

```
SELECT BusinessEntityID
FROM Purchasing.vVendorWithContacts
WHERE EmailAddress = 'cheryl1@adventure-works.com'

SELECT BusinessEntityID
FROM Purchasing.vVendorWithContacts
WHERE EmailAddress = 'stuart2@adventure-works.com'

SELECT BusinessEntityID
FROM Purchasing.vVendorWithContacts
WHERE EmailAddress = 'eunice0@adventure-works.com'
```

在执行完上面的查询之后，执行如下的查询：

```
SELECT   t.text,
      st.total_logical_reads
FROM sys.dm_exec_query_stats st
CROSS APPLY sys.dm_exec_sql_text(st.sql_handle) t
WHERE text LIKE '%Purchasing.vVendorWithContacts%'
```

这个查询返回：

```
text                                                    total_logical_reads
SELECT BusinessEntityID
FROM Purchasing.vVendorWithContacts
WHERE EmailAddress = 'stuart2@adventure-works.com'      12
SELECT BusinessEntityID
FROM Purchasing.vVendorWithContacts
WHERE EmailAddress = 'cheryl1@adventure-works.com'      12
SELECT BusinessEntityID
FROM Purchasing.vVendorWithContacts
WHERE EmailAddress = 'eunice0@adventure-works.com'      12
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意，尽管每个对Purchasing.vVendorWithContacts表的查询除了EmailAddress文本值外都是相同的，但还是为每个查询创建统计信息行。这就是没有预定义查询文本的应用程序产生的问题。

要解决这个问题，SQL Server 2008在sys.dm_exec_query_stats动态管理视图中引入了两个新列，分别是query_hash和query_plan_hash。这两列包含二进制散列值。Query_hash二进制值与那些除了文本值（在这个示例中，e-mail地址不同）外都相同的查询一样。生成的query_plan_hash二进制值与那些使用相同查询计划的查询一样。这两列增加了聚合整个跨越相同查询或查询执行计划的统计信息的能力：

```
SELECT  st.query_hash,
        COUNT(t.text) query_count,
        SUM(st.total_logical_reads) total_logical_reads
FROM sys.dm_exec_query_stats st
CROSS APPLY sys.dm_exec_sql_text(st.sql_handle) t
WHERE text LIKE '%Purchasing.vVendorWithContacts%'
GROUP BY st.query_hash
```

这个查询返回：

```
query_ hash             query_count     total_logical_reads
0x5C4B94191341266A      3               36
```

解析

这个技巧先执行3个除了在WHERE子句中EmailAddress列定义的文本值外都相同的查询。在这之后，演示了查询sys.dm_exec_query_stats动态管理视图来查看每个查询逻辑读的统计信息。每个对Purchasing.vVendorWithContacts的查询生成3个独立的行，而不是聚合的单行。如果尝试捕获TOP X数量的高资源使用查询，就会带来问题，因为结果可能不会反映出许多存在于查询计划缓存中的相同查询的变种。

为了解决这个问题，演示了使用SQL Server 2008中引入到sys.dm_exec_query_stats动态管理视图中的新列query_hash。

一点点地讲解这个查询，查询的SELECT子句引用这个新列query_hash，并为使用不同文本值的不同的查询生成COUNT，以及这些查询中逻辑读的SUM：

```
SELECT  st.query_hash,
        COUNT(t.text) query_count,
        SUM(st.total_logical_reads) total_logical_reads
```

FROM子句引用sys.dm_exec_query_stats动态管理视图，并使用CROSS APPLY来访问基于sql_handle的查询文本：

```
FROM sys.dm_exec_query_stats st
CROSS APPLY sys.dm_exec_sql_text(st.sql_handle) t
```

我将结果集减小到引用Purchasing.vVendorWithContacts视图的那些查询：

```
WHERE text LIKE '%Purchasing.vVendorWithContacts%'
```

最后，由于用query_hash聚合了统计信息，在query_hash列上使用GROUP BY子句：

```
GROUP BY st.query_hash
```

3个查询的Query_hash值0x5C4B94191341266A都是相同的，这样可以将每个不同的行聚合到单行中，并适当地合计感兴趣的统计信息列。用query_hash或queyr_plan_hash进行聚合有助于理解特定查询或计划模式及它们相关联的资源消耗。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

28.2.6　标识出最大瓶颈

你曾经是否遇到过客户或同事报告说“SQL Server运行缓慢”？当你询问更多细节时，那个人可能表达不清楚相关的性能问题，或可能会认为它们只是一些偶然事件。

在这种情况下，能用来标识并减小可能范围的首选工具就是sys.dm_os_wait_stats动态管理视图。这个DMV通过在SQL Server实例中执行线程，来提供所有发生的等待的运行总和。每次SQL Server重启之后，或如果你手工清空了统计信息，数据就会被清零并在SQL Server实例的正常运行时间进行累计。

SQL Server将这些等待分为几个不同的类型。一些类型只表示当线程处于等待时的平静期，而其他等待类型表示在特定资源上的外部或内部争用。

提示　这里描述的方法是等待和队列方法论中的一部分。关于这个方法论的更深入的讨论可以在技术白皮书http://technet.microsoft.com/en-us/sqlserver/bb331794.aspx中找到。

随后的技巧显示了SQL Server实例自从上一次清空或实例启动后累积最大的两个等待类型：

```
SELECT   TOP 2
   wait_type, wait_time_ms
FROM sys.dm_os_wait_stats
WHERE wait_type NOT IN
   ('LAZYWRITER_SLEEP', 'SQLTRACE_BUFFER_FLUSH',
    'REQUEST_FOR_DEADLOCK_SEARCH', 'LOGMGR_QUEUE',
    'CHECKPOINT_QUEUE', 'CLR_AUTO_EVENT','WAITFOR',
    'BROKER_TASK_STOP', 'SLEEP_TASK', 'BROKER_TO_FLUSH')
ORDER BY wait_time_ms DESC
```

这个查询返回如下内容（基于SQL Server活动的不同，你的结果将会不同）：

```
wait_type      wait_time_ms
LCK_M_U        31989
LCK_M_S        12133
```

在这个例子中，SQL Server的两个最长等待是关于请求等待获得更新锁和共享锁。可以通过在SQL Server联机丛书或在微软发布的等待和队列白皮书中查阅了解它们。在这个技巧的例子中，最长的两个等待类型是经常和长时间运行阻塞有关的。这表示如果应用程序遇到了性能问题，你会明智地想到使用更细颗粒的工具（动态管理视图，SQL Profiler）来提供额外的长时间运行阻塞的证据。查看sys.dm_os_wait_stats最主要的目的是对性能问题进行故障排除，而不相关的问题或其他低优先级的问题不是根本原因。

752　如果希望清空当前累积的等待类型统计信息，可以运行如下查询：

```
DBCC SQLPERF ('sys.dm_os_wait_stats', CLEAR)
```

清空等待类型统计信息允许你基于定义的时间段内提供累积的等待统计信息。

解析

这个技巧演示了使用sys.dm_os_wait_stats动态管理视图来帮助确定SQL Server实例中占主导地位的等待统计信息。

SELECT子句选择了等待类型和等待时间（以毫秒为单位）列：

```
SELECT TOP 2
   wait_type, wait_time_ms
FROM sys.dm_os_wait_stats
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

由于不是所有等待类型都会产生问题，使用WHERE子句筛选非外部或非资源等待（尽管这不是你想要滤除的等待类型的定义列表）：

```
WHERE wait_type NOT IN
   ('LAZYWRITER_SLEEP', 'SQLTRACE_BUFFER_FLUSH',
    'REQUEST_FOR_DEADLOCK_SEARCH', 'LOGMGR_QUEUE',
    'CHECKPOINT_QUEUE', 'CLR_AUTO_EVENT','WAITFOR',
   'BROKER_TASK_STOP', 'SLEEP_TASK', 'BROKER_TO_FLUSH')
ORDER BY wait_time_ms DESC
```

动态管理视图的数据以实例级别进行分组，而不是以数据库级别，因此这是性能问题故障排除任务的一个好的开始。这不是最终的解决方案，但是它是能帮助你在对不良定义的性能问题进行故障排除时定位正确方向的非常有用的工具。对于随着时间的推移而确立的趋势，这个动态管理视图也会派上用场。如果出现了新的等待类型，这可能就是新性能问题的重要指标。

28.2.7 通过数据库和文件标识出 I/O 争用

假定你查询sys.dm_os_wait_stats并发现等待大多数是由于I/O争用引起的。由于等待状态被限定在SQL Server实例级别的范围，下一步就是标识出哪一个数据库遭受最高数量的I/O争用。

sys.dm_io_virtual_file_stats动态管理视图（DMV显示的数据与fn_virtualfilestats函数相同）可以用来确定哪一个数据库具有最高数量的读取、写入和I/O阻塞行为。

这个技巧演示了查看数据库I/O统计信息，并且以I/O阻塞排序，I/O阻塞以毫秒为单位，并表示自从实例最后重启或数据库创建后用户等待在文件中完成读取或写入的I/O操作所用的总时间：

```
SELECT DB_NAME(database_id) DatabaseNM,
       file_id FileID,
       io_stall IOStallsMs,
       size_on_disk_bytes FileBytes,
       num_of_bytes_written BytesWritten,
       num_of_bytes_read BytesRead
FROM sys.dm_io_virtual_file_stats(NULL, NULL)
ORDER BY io_stall DESC
```

753

这个查询返回：

DatabaseNM	FileID	IOStallsMs	FileBytes	BytesWritten	BytesRead
tempdb	1	3729468	92602368	216678400	78725120
AdventureWorks	1	520481	405536768	234594304	328687616
AdventureWorks	2	54145	18874368	279374848	409600
master	1	9927	4194304	2998272	22315008
msdb	1	4435	12124160	843776	20725760
tempdb	2	2095	1048576	4382720	663552
master	2	997	786432	1622016	458752
model	1	456	1310720	57344	4431872
msdb	2	182	524288	262144	450560
AdventureWorks	3	150	1048576	139264	196608
AdventureWorks	7	96	1048576	90112	131072
AdventureWorks	4	93	1048576	139264	196608
model	2	66	524288	61440	450560
AdventureWorks	5	24	1048576	139264	196608
AdventureWorks	6	17	1048576	139264	131072

解析

这个技巧演示了使用sys.dm_io_virtual_file_stats动态管理视图，来返回每个数据库和SQL

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Server实例中文件的统计信息。DMV有两个输入参数：第一个是数据库ID，第二个是文件ID。为数据库ID指定NULL，即结果显示所有的数据库。为文件ID指定NULL，即结果显示数据库所有的文件。

在这个技巧中，指定返回所有数据库和相关文件：

```
FROM sys.dm_io_virtual_file_stats(NULL, NULL)
```

我还以I/O阻塞进行降序排序，先来查看最高I/O延迟活动的文件：

```
ORDER BY io_stall DESC
```

这些结果标识出AdventureWorks数据库中ID为2文件（在这个示例中为日志文件）中的最高数量的阻塞。如果你已经确定I/O是主要的性能问题，则使用sys.dm_io_virtual_file_stats是一个高效的方法，有助于减少故障排除工作中要关注的数据库和文件。

28.3　索引调优

下一组技巧展示了管理索引的方法。先讲一下如何去做下面的事情。

- 标识出索引碎片，这样就可以指出应该对哪个索引进行重新生成或重新组织。
- 显示索引的使用情况，因此可以判断出哪一个索引没有被SQL Server使用。

在开始这个技巧之前，我想花一分钟时间讨论一些常规的索引最佳经验。在考虑这些最佳经验时，
754 要记住像查询调优一样，几乎没有严格的“总是”或“从不”规则。SQL Server中索引的使用依赖于许多因素，包括（但不仅限于）查询的结构、查询中引用的表、引用的列、表的行数、索引列中数据的唯一性。当建立索引策略时要牢记如下几个基本的指导方针。

- 基于高优先级和使用频繁的查询增加索引。基于你的业务需求，提前确定可接受的查询执行时间可能是多少。
- 不要同时增加多个索引。而应该增加一个索引同时测试查询，去看新的索引是否被使用。如果没有被使用，则删除它。如果被使用了，则对它进行测试，以确保它不会对其他查询产生负面影响。请记住，每个附加的索引都会在基础表的数据修改上增加额外的开销。
- 总是在每个表中都增加一个聚集索引，除非你有非常好的原因不这样做。没有聚集索引的表称为堆，意思是其中存储的数据没有特定的顺序。在索引重新生成或重新组织期间，聚集索引依照聚集键和它重排序的数据页进行排序。可是堆不会在索引重新生成或重新组织期间被重新生成，所以会脱离控制地生长，占用的数据页比必要的多很多。
- 随时监视查询性能。随着数据和应用程序活动的改变，索引的性能和效能也会改变。
- 由于返回查询的结果需要更多的I/O操作，碎片化的索引会降低查询性能。基于时间表或需要，通过重新生成和/或重新组织你的索引保持索引碎片最小化。
- 选择很少改变、高度唯一、数据类型宽度狭窄的列作为聚集索引键。宽度非常重要，因为所有非聚集索引也包含了聚集索引键。聚集索引在应用到使用了范围查询的列中时是有用的。这包括使用了运算符BETWEEN、>、>=、<和<=的查询。对于返回大量结果集或高度依赖ORDER BY和GROUP BY子句的查询，聚集索引键也有助于减少执行时间。
- 非聚集索引对于小的或只有一行的结果集是理想的选择。再一次提醒，应该基于查询中的应用挑选列，特别在JOIN和WHERE子句中时。非聚集索引应该建立在包含高度唯一数据的列上。就像在第5章讨论到的，不要忘记为非键列考虑使用覆盖查询和SQL Server引入的INCLUDE新功能。
- 为那些定位在只读文件组或数据库上的索引使用100%填充因子。因为完成查询的结果集需要较少的数据页，所以这减少了I/O操作并且可以提升查询性能。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 基于你使用的查询设法预估需要哪些索引——但是也不要害怕频繁地使用数据库引擎优化顾问工具。使用数据库引擎优化顾问工具，SQL Server可以估计你的查询或查询的批处理，然后为了帮助查询运行得更快决定应该增加（或删除）哪一个索引。我将在稍后展示它。

下一技巧将展示如何显示索引碎片。

755

28.3.1 显示索引碎片

碎片是表中数据修改自然产生的副产品。当在数据库中更新数据时，索引（基于索引键）的逻辑顺序不和实际数据页的物理顺序同步了。当数据页变得越来越无序，为了返回查询请求的结果，需要更多的I/O操作。重新生成或重新组织一个索引允许通过同步逻辑索引顺序、重排列物理数据页去匹配逻辑索引顺序，来整理索引碎片。

注解 *索引管理的介绍请看第5章，索引碎片整理和重新组织的介绍请看第23章。*

动态管理函数sys.dm_db_index_physical_stats返回能确定索引碎片级别的信息。

sys.dm_db_index_physical_stats的语法如下：

```
sys.dm_db_index_physical_stats (
    { database_id | NULL }
    , { object_id | NULL }
    , { index_id | NULL | 0 }
    , { partition_number | NULL }
    , { mode | NULL | DEFAULT }
)
```

表28-1描述了这个命令的参数。

表28-1 sys.dm_db_index_physical_stats的参数

参 数	描 述
database_id \| NULL	这个参数定义要检测索引的数据库ID。如果为NULL，返回SQL Server实例中所有数据库的信息
object_id \| NULL	这个参数指定要检测的表和视图（索引视图）的对象ID。如果为NULL，返回所有表的信息
index_id \| NULL \| 0	这个参数为要检测的指定索引ID。如果为NULL，返回表中所有索引的信息
partition_number \| NULL	这个参数定义要检测的分区的指定分区编号。如果为NULL，返回基于已定义数据库/表/选择的索引的所有分区信息
LIMITED \| SAMPLED \| DETAILED \| NULL \| DEFAULT	这些模式影响了如何收集碎片数据。LIMITED模式扫描堆所有的页，但对于索引，则只扫描叶级上面的父级别页。SAMPLED收集在堆或索引中1%采样率的数据。DETAILED模式扫描所有页（堆或索引）。DETAILED是执行最慢的，但也是最精确的选项。指定NULL或DEFAULT的效果与LIMITED模式的相同

在这个示例中，查询动态管理视图sys.dm_db_index_physical_stats检索AdventureWorks数据库中平均碎片率大于30的所有对象：

756


```
USE AdventureWorks
GO

SELECT OBJECT_NAME(object_id) ObjectName,
       index_id,
       index_type_desc,
       avg_fragmentation_in_percent
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FROM sys.dm_db_index_physical_stats
(DB_ID('AdventureWorks'),NULL, NULL, NULL, 'LIMITED')
WHERE avg_fragmentation_in_percent > 30
ORDER BY OBJECT_NAME(object_id)
```

这个查询返回下面的（部分）结果：

```
ObjectName              index_id   index_type_desc        avg_fragmentation_in_percent
BillOfMaterials         2          NONCLUSTERED INDEX     33.3333333333333
BusinessEntityContact   1          CLUSTERED INDEX        50
BusinessEntityContact   2          NONCLUSTERED INDEX     50
BusinessEntityContac t  3          NONCLUSTERED INDEX     50
BusinessEntityContact   4          NONCLUSTERED INDEX     50
CountryRegion           1          CLUSTERED INDEX        50
DatabaseLog             0          HEAP                   32.6732673267327
...
```

第二个示例返回指定数据库、表和索引的碎片：

```
SELECT OBJECT_NAME(f.object_id) ObjectName,
       i.name IndexName,
       f.index_type_desc,
       f.avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats
         (DB_ID('AdventureWorks'),
         OBJECT_ID('Production.ProductDescription'),
         2,
         NULL,
         'LIMITED') f
INNER JOIN sys.indexes i ON
   i.object_id = f.object_id AND
   i.index_id = f.index_id
```

这个查询返回：

```
ObjectName          IndexName                  index_type_desc       avg_fragmentationin_percent
ProductDescription  AK_ProductDescription_     NONCLUSTERED INDEX    66.6666666666667
                    rowguid
```

解析

第一个例子首先将数据库上下文改为AdventureWorks数据库：

```
USE AdventureWorks
GO
```

757 由于OBJECT_NAME函数是数据库上下文可识别的，改变数据库上下文确保你查看到正确的对象名称。下一步，SELECT子句显示对象名称、索引ID、描述和平均的碎片百分比：

```
SELECT OBJECT_NAME(object_id) ObjectName,
       index_id, index_type_desc,
       avg_fragmentation_in_percent
```

列index_type_desc告诉你索引是堆、聚集索引、非聚集索引、主XML索引还是辅助XML索引。

下一步，FROM子句引用了sys.dm_db_index_physical_stats目录函数。括号中为参数，包括数据库名、扫描模式，其他参数为NULL：

```
FROM sys.dm_db_index_physical_stats
(DB_ID('AdventureWorks'),NULL, NULL, NULL, 'LIMITED')
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

由于sys.dm_db_index_physical_stats像表一样被引用（不像2000版本中的DBCC SHOWCONTIG），因此使用WHERE子句来限定只在结果中返回碎片百分比大于等于31%的行：

```
WHERE avg_fragmentation_in_percent > 30
```

查询返回了AdventureWorks数据库中对象的碎片大于30%的几行。列avg_fragmentation_in_percent显示聚集索引或非聚集索引的逻辑碎片，返回索引的叶级无序页的百分比。对于堆来说，avg_fragmentation_in_percent显示区级碎片。关于区，回忆一下SQL Server在页级读写数据。存储在块中的页被称为区，它由8个连续的8KB的页组成。使用avg_fragmentation_in_percent，你可以决定是否需要通过ALTER INDEX对索引重新生成或重新组织。

在第二个示例中，显示指定数据库、表和索引的碎片。SELECT子句包含了对索引名的引用（不是索引编号）：

```
SELECT OBJECT_NAME(f.object_id) ObjectName,
       i.name IndexName,
       f.index_type_desc,
       f.avg_fragmentation_in_percent
```

FROM子句包含了指定的表名，它通过使用OBJECT_ID函数转换为ID。第三个参数包含了要检测碎片的索引的索引编号：

```
FROM sys.dm_db_index_physical_stats
(DB_ID('AdventureWorks'),
OBJECT_ID('Production.ProductDescription'),
2,
NULL,
'LIMITED') f
```

通过object_id和index_id，系统目录视图sys.indexes联结了函数sys.dm_db_index_physical_stats。

```
INNER JOIN sys.indexes i ON
   i.object_id = f.object_id AND
   i.index_id = f.index_id
```

查询只返回了指定索引的碎片结果。

758

28.3.2 显示索引使用情况

在数据库中创建有用的索引需要考虑数据库读写性能。在加速SELECT查询的同时，索引减慢了数据的修改。必须平衡读取与数据修改带来的索引开销的代价 / 受益。每一个增加到表中的索引都会以数据修改速度为代价改善查询性能。在这之上，索引的效能会随着数据的改变而改变，由此几个星期前索引还是有用的，今天可能就没用了。如果想在表中创建索引，则它们应该在高优先级的查询中很好地使用。

可以通过查询动态管理视图sys.dm_db_index_usage_stats确定未使用的索引。这个视图从SQL Server实例上次重启以后返回有关索引查找、扫描、更新或查找的次数的统计信息。它也返回引用索引的最后时间。

在这个示例中，查询动态管理视图sys.dm_db_index_usage_stats去查看表Sales.Customer中的索引是否被使用过。在引用sys.dm_db_index_usage_stats之前，要在表Sales.Customer上执行两个查询，第一个返回所有的行和列，第二个返回具有指定的TerritoryID的AccountNumber列：

```
SELECT *
FROM Sales.Customer
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SELECT AccountNumber
FROM Sales.Customer
WHERE TerritoryID = 4
```

在执行完查询之后，查询动态管理视图sys.dm_db_index_usage_stats：

```
SELECT i.name IndexName, user_seeks, user_scans,
last_user_seek, last_user_scan
FROM sys.dm_db_index_usage_stats s
INNER JOIN sys.indexes i ON
   s.object_id = i.object_id AND
   s.index_id = i.index_id
WHERE database_id = DB_ID('AdventureWorks') AND
     s.object_id = OBJECT_ID('Sales.Customer')
```

这个查询返回：

IndexName	user_seeks	user_scans	last_user_seek	last_user_scan
IX_Customer_TerritoryID	1	0	2008-10-15 17:13:35.487	NULL
PK_Customer_CustomerID	0	1	NULL	2008-10-15 17:13:34.237

解析

动态管理视图sys.dm_db_index_usage_stats允许查看SQL Server实例使用了哪个索引。在最后一次SQL Server重启之后统计信息才生效。

在这个技巧中，在表Sales.Customer上执行两个查询。在执行完查询后，查询动态管理视图sys.dm_db_index_usage_stats。

SELECT子句显示了索引的名称、用户查找和用户扫描的次数以及用户最后查找和最后扫描的时间：

```
SELECT i.name IndexName, user_seeks, user_scans,
last_user_seek, last_user_scan
```

759

FROM子句通过object_id和index_id将动态管理视图sys.dm_db_index_usage_stats联结到系统目录视图sys.indexes上（所以索引名应该显示在结果中）：

```
FROM sys.dm_db_index_usage_stats s
INNER JOIN sys.indexes i ON
   s.object_id = i.object_id AND
   s.index_id = i.index_id
```

WHERE子句限定只显示AdventureWorks数据库上的索引以及表Sales.Customer上的索引。DB_ID函数用来返回数据库系统ID，OBJECT_ID函数用来得到表的对象ID：

```
WHERE database_id = DB_ID('AdventureWorks') AND
     s.object_id = OBJECT_ID('Sales.Customer')
```

查询返回两行，显示表Sales.Customer的聚集索引PK_Customer_CustomerID最近被访问过（很可能是通过第一个SELECT *查询），并且第二个查询（限定了TerritoryID = 4）使用了非聚集索引IX_Customer_TerritoryID。

索引帮助提高了查询性能，但也增加了磁盘空间和数据修改的开销。使用动态管理视图sys.dm_db_index_usage_stats，可以监视索引是否真的被使用，如果没有，则把它们替换为更有效的索引。

28.4　统计信息

正如在第22章中讨论的，数据库选项AUTO_CREATE_STATISTICS使SQL Server能自动产生关于列中数

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

据分布的统计信息。数据库选项AUTO_UPDATE_STATISTICS自动更新既有的表或索引视图的统计信息。由于这个选项对查询性能会产生关键性的影响，除非有非常特殊的原因，否则一定不要在数据库中关闭这个选项。

统计信息对于有效的查询执行和查询性能来说是至关重要的，它允许SQL Server在产生执行计划时选择适当的物理操作。SQL Server可以手动或自动地产生表和索引视图的统计信息，收集产生有效查询执行计划的信息。

下面的几个技巧将展示如何直接使用统计信息。在阅读这些技巧时请记住，只要可能，要让SQL Server管理数据库中统计信息的自动创建和更新。这样会在特殊的故障诊断过程或做了重大数据改变时（例如，在大量数据载入后马上执行sp_updatestats）省去执行大部分这些命令。

28.4.1 手动创建统计信息

SQL Server通常基于查询活动产生所需要的统计信息。然而，如果你希望在列上显式地创建统计信息，可以使用CREATE STATISTICS命令。

语法如下：

```
CREATE STATISTICS statistics_name
ON { table | view } ( column [ ,...n ] )
    [ WITH
        [ [ FULLSCAN
          | SAMPLE number { PERCENT | ROWS }
          | STATS_STREAM = stats_stream ] [ , ] ]
    [ NORECOMPUTE ]
]
```

760

表28-2描述了这个命令的参数。

表28-2 CREATE STATISTICS参数

参 数	描 述
statistics_name	这个参数定义新统计信息的名称
table \| view	这个参数指定要在其中创建统计信息的表或索引视图
column [,...n]	这个参数指定用于产生统计信息的一列或多列
FULLSCAN \| SAMPLE number {PERCENT \| ROWS}	当指定FULLSCAN时，在产生统计信息时读取所有行。SAMPLE可以读取指定数量的行或读取指定百分比的行
STATS_STREAM = stats_stream	此语法仅供微软内部使用
NORECOMPUTE	这个选项指定，一旦创建了统计信息，即使数据发生改变，统计信息也不会被更新。建议限制使用这个选项。及时的统计信息会帮助SQL Server产生理想的查询计划

在这个示例中，在列Sales.Customer CustomerType上创建新的统计信息：

```
CREATE STATISTICS Stats_Customer_AccountNumber
ON Sales.Customer (AccountNumber)
WITH FULLSCAN
```

解析

这个技巧展示了在表Sales.Customer上手动创建统计信息。代码的第一行指定了统计信息的名称：

```
CREATE STATISTICS Stats_Customer_AccountNumber
```

代码的第二行指定了要在其中创建统计信息的表，随后是用于产生统计信息的列名：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ON Sales.Customer (AccountNumber)
```

代码的最后一行指定了为了生成统计信息要读取表中所有的行：

```
WITH FULLSCAN
```

28.4.2　行子集上创建统计信息

在5.4.3节的技巧中，演示了覆盖小百分比的行的筛选非聚集索引的能力。这样做可以减小索引的大小，并只需读取索引实体一小部分来提升查询的性能，否则不得不全部处理索引实体。创建筛选索引也会创建相关的统计信息。这些统计信息使用相同的筛选谓词并且由于以较小的行集作为采样，结果会更加精确。

761

还可以显式地使用CREATE STATISTICS命令创建筛选统计信息。与创建筛选索引相似，筛选统计信息也支持几个使用比较运算符的筛选谓词，包括IS、IS NOT、=、<>、>、<等。

如下查询演示在Sales.SalesOrderDetail表中UnitPrice列的范围值上创建筛选统计信息：

```
CREATE STATISTICS Stats_SalesOrderDetail_UnitPrice_Filtered
ON Sales.SalesOrderDetail (UnitPrice)
WHERE UnitPrice >= 1000.00 AND
      UnitPrice <= 1500.00
WITH FULLSCAN
```

解析

这个技巧演示了创建筛选统计信息。与筛选索引相似，我在CREATE STATISTICS调用的定义中添加WHERE子句。在列上创建统计信息，将生成最多200个区间值的柱状图（指定每个区间值内有多少行，以及比当前键小但比以前的键少的行数[①]）。查询优化器基于高精确度的统计信息进行处理。筛选统计信息允许指定应用程序关注到值的关键范围，为数据的子集产生更精确的统计信息。

28.4.3　更新统计信息

在创建了统计信息之后，如果希望手动更新统计信息，可以使用UPDATE STATISTICS命令。

语法如下：

```
UPDATE STATISTICS table | view
    [
        {
            { index | statistics_name }
          | ( { index |statistics_name } [ ,...n ] )
                }
    ]
    [   WITH
        [
            [ FULLSCAN ]
            | SAMPLE number { PERCENT | ROWS }
            | RESAMPLE
        ]
         [ , ] [ ALL | COLUMNS | INDEX ]
        [ [ , ] NORECOMPUTE ]
]
```

762

表28-3描述了这个命令的参数。

① 意思是说，使用了筛选统计信息之后，查询效率更高，即可以比未使用这个功能之前检索更少的行。——译者注

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表28-3 UPDATE STATISTICS参数

参 数	描 述
table \| view	这个参数定义要更新统计信息的表名或索引视图名
{ index \| statistics_name }]	这个参数指定要更新的索引名称或统计信息名称
FULLSCAN] SAMPLE number { PERCENT \| ROWS } \| RESAMPLE	当指定FULLSCAN时，在生成统计信息时读取所有行。SAMPLE读取指定数量的行或指定百分比的行。RESAMPLE基于原始抽样方法更新统计信息
[ALL \| COLUMNS \| INDEX]	当指定ALL时，更新所有既有的统计信息。当指定COLUMN时，只更新列的统计信息。当指定INDEX时，只更新索引统计信息
NORECOMPUTE	这个选项指定，一旦创建了统计信息，即便数据发生了改变，也不更新统计信息。在这里要重申一下，建议限制使用这个选项。及时的统计信息会帮助SQL Server产生理想的查询计划

这个示例更新了在前面技巧中为表Sales.Customer创建的统计信息，基于最新的数据刷新统计信息：

```
UPDATE STATISTICS Sales.Customer
WITH FULLSCAN
```

解析

这个示例更新了在前面技巧中为表Sales.Customer创建的统计信息，并用最新的数据刷新统计信息。代码的第一行指定了包含要更新的统计信息的表名：

```
UPDATE STATISTICS Sales.Customer
```

代码的最后一行指定了为了更新统计信息读取表中所有行：

```
WITH FULLSCAN
```

28.4.4 生成及更新所有表的统计信息

通过系统存储过程sp_createstats，也可以自动生成数据库中所有那些列上没有关联统计信息的表的统计信息。

语法如下：

```
sp_createstats [ [ @indexonly = ] 'indexonly' ]
       [ , [ @fullscan = ] 'fullscan' ]
   [ , [ @norecompute = ] 'norecompute' ]
```

表28-4描述了这个命令的参数。

763

表28-4 sp_createstats的参数

参 数	描 述
indexonly	当指定了indexonly时，只考虑在那些使用了索引的列上创建统计信息
fullscan	当指定了fullscan，对所有行求值来生成统计信息。如果没有指定，默认的行为是通过采样提取统计信息
norecompute	norecompute选项指定，一旦创建了统计信息，即便数据发生了改变，也不更新统计信息。像CREATE STATISTICS和UPDATE STATISTICS一样，建议限制使用这个选项。及时的统计信息会帮助SQL Server产生理想的查询计划

这个示例展示了对数据库中那些没有关联统计信息的列创建新的统计信息：

```
EXEC sp_createstats
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

它返回如下的部分结果集：

```
表'AdventureWorks.Production.ProductProductPhoto':
正为以下列创建统计信息:
    Primary
    ModifiedDate
表'AdventureWorks.Sales.StoreContact':
正为以下列创建统计信息:
    ModifiedDate
表'AdventureWorks.Person.Address':
正为以下列创建统计信息:
    AddressLine2
    City
    PostalCode
    ModifiedDate
...
```

如果希望在当前数据库中更新所有的统计信息，可以使用系统系统存储过程sp_updatestats。该存储过程只更新必要的（当数据发生改变时）统计信息。不会更新未改变数据的统计信息。

下一个示例自动地更新在当前数据库中的所有统计信息：

```
EXEC sp_updatestats
GO
```

它返回如下的部分结果。注意“不需要更新……”这个信息。你看到的结果可能会因表统计信息的状态不同而不同：

```
正在更新 [Production].[ProductProductPhoto]
    [PK_ProductProductPhoto_ProductID_ProductPhotoID], 不需要更新...
    [AK_ProductProductPhoto_ProductID_ProductPhotoID], 不需要更新...
    [_WA_Sys_00000002_01142BA1], 不需要更新...
    [Primary], 不需要更新...
    [ModifiedDate], 不需要更新...
    已更新0条索引/统计信息，5条不需要更新。
...
```

764

28.4.5　查看详细的统计信息

可以使用DBCC SHOW STATISTICS命令查看详细的统计信息。

它的语法如下：

```
DBCC SHOW_STATISTICS ( 'table_name' | 'view_name' , target )
[ WITH [ NO_INFOMSGS ]
< STAT_HEADER | DENSITY_VECTOR | HISTOGRAM > [ , n ] ]
```

表28-5描述了这个命令的参数。

表28-5　DBCC SHOW_STATISTICS参数

参　数	描　述
'table_name' \| 'view_name'	这个参数定义要查看的表或索引视图
target	这个参数指定要查看的索引或统计信息的名称
NO_INFOMSGS	当指定NO_INFOMSGS时，取消信息性消息
STAT_HEADER \| DENSITY_VECTOR \| HISTOGRAM [,n]	如果指定其中一个或多个选项，可限制该命令返回的结果集。不指定任何选项表示将返回这3种结果集

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个示例展示了如何查看表Sales.Customer的统计信息Stats_Customer_AccountNumber的统计信息内容：

```
DBCC SHOW_STATISTICS ( 'Sales.Customer' , Stats_Customer_AccountNumber)
```

它返回如下的结果集：

Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows
Stats_Customer_AccountNumber	Mar 30 2008 12:49PM	19820	19820	152	1	10	YES	NULL	19820

All density	Average Length	Columns
5.045409E-05	10	AccountNumber

RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
AW00000001	0	1	0	1
...				
AW00027042	127	1	127	1
AW00027298	255	1	255	1
AW00027426	127	1	127	1
...				
AW00030118	0	1	0	1

765

解析

这个技巧演示了如何获取更多关于列的统计信息的指定信息。在这个技巧示例的结果中，列All density表示列的选择性（selectivity）。选择性是指给定具体列的值返回的行的百分比。较低的All density值表示较高的选择性。经常用高选择性的列作为有用的索引（对查询优化过程是有用的）。

在这个技巧的示例中，All density值为5.045409E-05（浮点型），等于十进制值0.00005045409。这是我除以行数的结果，在这个例子中行数是19 820。

如果有高密度的相似值，并且低选择性（返回的多个行中很可能为同一值）的列，你可以做一个深思熟虑的假设，在这个特定列上的索引对SQL Server生成查询执行计划不太可能非常有用。

28.4.6 删除统计信息

要删除统计信息，使用DROP STATISTICS命令。

语法如下：

```
DROP STATISTICS table.statistics_name | view.statistics_name [ ,...n ]
```

使用这个命令可以删除一个或多个统计信息，需要加上表或索引视图名的前缀。

在这个示例中，从数据库中删除统计信息Sales.Customer.Stats_Customer_AccountNumber：

```
DROP STATISTICS Sales.Customer.Stats_Customer_AccountNumber
```

解析

这个技巧通过使用DROP STATISTICS删除用户创建的统计信息。统计信息使用3部分组成的名称schema.table.statistics_name。

28.5 杂项技术

28

下面的几个技巧详细地讲述了一些在本章前面没有提到的技术。这些技巧将演示下面一些内容：

- 用sp_executesql系统存储过程来替代动态SQL和动态存储过程；

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 强制查询使用指定的查询计划；
- 使用计划指南不实际修改应用程序的SQL代码为既有的查询应用查询提示；
- 基于指向缓存计划的指针创建计划指南；
- 检查计划指南的有效性（在引用对象移交到无效计划的情况下）；
- 强制非参数化的查询参数化；
- 使用新的SQL Server 2008资源调控器特性来限制查询资源消耗（对于CPU和内存）。

这部分先描述使用动态SQL的替代方法。

766

28.5.1　使用动态 SQL 的替代方法

使用EXECUTE命令，可以执行在批处理、存储过程或函数中由字符串组成的内容。也可以使用EXECUTE的缩写EXEC。

例如，随后的语句执行了从表Sales.Currency中SELECT的操作：

```
EXEC ('SELECT CurrencyCode FROM Sales.Currency')
```

尽管这个技术允许动态格式化可以执行的字符串，它还是有很大的风险。首先，最大的风险是SQL注入带来的危险。SQL注入可以将恶意代码插入到以后将传递给SQL Server供分析和执行的字符串中。允许用户输入连接到SQL字符串并执行的变量，会对数据库引发各种各样的危险（更不用说潜在的保密性问题了）。这段恶意代码如果执行在具有充分权限的上下文中，它可以删除表、读取敏感数据甚至是关闭SQL Server进程。

字符串执行技术的第二个问题是它们的性能。尽管动态生成SQL有时可能性能还不错，但查询的性能也是不可靠的。不像存储过程，动态生成的SQL、常规即席SQL批处理及语句将会让SQL Server在它们每次执行时生成新的执行计划。

如果应用程序没办法使用存储过程，作为替代方法，系统存储过程sp_executesql通过创建和使用可重用的、只有查询参数发生更改的查询执行计划处理动态SQL的性能问题。参数是类型安全的，意思是不能以非指定的数据类型使用它们。当需要在即席语句和存储过程中做出选择时，这是不错的解决方案。

警告　sp_executesql解决了一些性能问题，但是不能完美地解决SQL注入问题。要小心连接到SQL字符串中的用户传递的参数！下面就会描述参数的功能。

sp_executesql的语法如下：

```
sp_executesql [ @stmt = ] stmt
[
    {, [@params=] N'@parameter_name data_type [ OUT | OUTPUT ][,...n]' }
     {, [ @param1 = ] 'value1' [ ,...n ] }
]
```

表28-6描述了这个命令的参数。

表28-6　sp_executesql参数

参　数	描　述
stmt	要执行的字符串
@parameter_name data_type[OUTPUT][,...n]	包含在字符串语句中的一个或多个参数。OUTPUT与存储过程的参数OUTPUT相似
'value1' [,...n]	传递到参数的实际值

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在这个示例中，通过指定ProductID、TransactionType以及Quantity的最小值查询表Production.TransactionHistoryArchive：

```
EXECUTE sp_executesql
N'SELECT TransactionID, ProductID,
         TransactionType, Quantity
FROM     Production.TransactionHistoryArchive
WHERE    ProductID = @ProductID AND
         TransactionType = @TransactionType AND
         Quantity > @Quantity',
         N'@ProductID int,
           @TransactionType char(1),
           @Quantity int',
              @ProductID =813,
              @TransactionType = 'S',
                 @Quantity = 5
```

查询返回如下的结果：

TransactionID	ProductID	TransactionType	Quantity
28345	813	S	7
31177	813	S	9
35796	813	S	6
36112	813	S	7
40765	813	S	6
47843	813	S	7
69114	813	S	6
73432	813	S	6

解析

系统存储过程sp_executesql允许执行一个动态生成的Unicode字符串。这个系统存储过程允许使用参数，这些参数又允许SQL Server重用它执行生成的查询执行计划。

注意在这个技巧中，由于sp_executesql需要Unicode语句字符串[①]，第一个参数之前加上了N'Unicode前缀。第一个参数也包括了SELECT查询本身，包含了WHERE子句中的参数：

```
EXECUTE sp_executesql
N'SELECT TransactionID, ProductID,
TransactionType, Quantity
FROM Production.TransactionHistoryArchive
WHERE ProductID = @ProductID AND
             TransactionType = @TransactionType AND
             Quantity > @Quantity',
```

第二个参数进一步定义了包含在第一个参数中的SQL语句的每个参数的数据类型。用逗号分隔每个参数：

```
N'@ProductID int,
@TransactionType char(1),
@Quantity int',
```

最后一个参数为所有包含的参数赋值，它在执行期间被动态地放入查询。

```
@ProductID =813,
@TransactionType = 'S',
@Quantity = 5
```

① sp_executesql也可以使用非Unicode字符串。——译者注

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

指定了3个参数的查询返回了8行数据。如果查询只改变不同的参数值再次执行，SQL Server很可能使用原始的查询执行计划（而不是创建新的执行计划）。

28.5.2　强制 SQL Server 使用查询计划

使用USE PLAN命令可以强制查询优化器为SELECT查询使用既有的特定查询计划。在一些罕见的情况下，SQL Server选择低效查询计划而不选择高效的，这时可以使用这个功能覆盖SQL Server的选择。与计划指南（稍后讲解）相似，应该只有那些有经验的SQL Server专业人员才可以使用这个选项，因为SQL Server查询优化器在决定是否重用或创建新查询执行计划时，通常情况下都会生成好的决策。

USE PLAN的语法如下：

```
USE PLAN N'xml_plan'
```

xml_plan参数是存储的查询执行计划的XML数据类型表现形式。可以使用几种方法来得到特定XML查询计划，其中包括SET SHOWPLAN_XML、SET STATISTICS XML、sys.dm_exec_query_plan动态管理视图、sys.dm_exec_text_query_plan和通过SQL Profiler的Showplan XML事件。

在这个示例中，使用SET STATISTICS XML提取出XML格式的查询计划，以用在USE PLAN命令中：

```
SET STATISTICS XML ON

SELECT TOP 10 Rate
FROM HumanResources.EmployeePayHistory
ORDER BY Rate DESC

SET STATISTICS XML OFF
```

将从SET STATISTICS XML返回的XMLDocument结果复制到下一个查询中。注意，在XML文档中所有的单引号（'）都必须通过一个额外的单引号进行转义（除了USE PLAN使用的之外）：

```
SELECT TOP 10 Rate
FROM HumanResources.EmployeePayHistory
ORDER BY Rate DESC
OPTION (USE PLAN
'<ShowPlanXML xmlns="http://schemas.microsoft.com/sqlserver/2004/07/showplan"
Version="1.1" Build="10.0.1424.2">
  <BatchSequence>
    <Batch>
      <Statements>
        <StmtSimple StatementText="SELECT TOP 10 Rate&#xD;&#xA;FROM
HumanResources.EmployeePayHistory&#xD;&#xA;ORDER BY Rate DESC&#xD;&#xA;&#xD;"
StatementId="1" StatementCompId="2" StatementType="SELECT"
StatementSubTreeCost="0.019825" StatementEstRows="10" StatementOptmLevel="TRIVIAL">
          <StatementSetOptions QUOTED_IDENTIFIER="false" ARITHABORT="true"
CONCAT_NULL_YIELDS_NULL="false" ANSI_NULLS="false" ANSI_PADDING="false"
ANSI_WARNINGS="false" NUMERIC_ROUNDABORT="false" />
          <QueryPlan DegreeOfParallelism="0" MemoryGrant="1024" CachedPlanSize="8"
CompileTime="20" CompileCPU="3" CompileMemory="72">
          <RelOp NodeId="0" PhysicalOp="Sort" LogicalOp="TopN Sort"
EstimateRows="10" EstimateIO="0.0112613" EstimateCPU="0.00419345" AvgRowSize="15"
EstimatedTotalSubtreeCost="0.019825" Parallel="0" EstimateRebinds="0"
EstimateRewinds="0">
              <OutputList>
                <ColumnReference Database="[AdventureWorks]"
Schema="[HumanResources]" Table="[EmployeePayHistory]" Column="Rate" />
              </OutputList>
              <MemoryFractions Input="1" Output="1" />
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
              <RunTimeInformation>
                <RunTimeCountersPerThread Thread="0" ActualRows="10"
ActualRebinds="1" ActualRewinds="0" ActualEndOfScans="1" ActualExecutions="1" />
              </RunTimeInformation>
              <TopSort Distinct="0" Rows="10">
                <OrderBy>
                  <OrderByColumn Ascending="0">
                    <ColumnReference Database="[AdventureWorks]"
Schema="[HumanResources]" Table="[EmployeePayHistory]" Column="Rate" />
                  </OrderByColumn>
                </OrderBy>
                <RelOp NodeId="1" PhysicalOp="Clustered Index Scan"
LogicalOp="Clustered Index Scan" EstimateRows="316" EstimateIO="0.00386574"
EstimateCPU="0.0005046" AvgRowSize="15" EstimatedTotalSubtreeCost="0.00437034"
TableCardinality="316" Parallel="0" EstimateRebinds="0" EstimateRewinds="0">
                  <OutputList>
                    <ColumnReference Database="[AdventureWorks]"
Schema="[HumanResources]" Table="[EmployeePayHistory]" Column="Rate" />
                  </OutputList>
                  <RunTimeInformation>
                    <RunTimeCountersPerThread Thread="0" ActualRows="316"
ActualEndOfScans="1" ActualExecutions="1" />
                  </RunTimeInformation>
                  <IndexScan Ordered="0" ForcedIndex="0" NoExpandHint="0">
                    <DefinedValues>
                      <DefinedValue>
                        <ColumnReference Database="[AdventureWorks]"
Schema="[HumanResources]" Table="[EmployeePayHistory]" Column="Rate" />
                      </DefinedValue>
                    </DefinedValues>
                    <Object Database="[AdventureWorks]" Schema="[HumanResources]"
Table="[EmployeePayHistory]"
Index="[PK_EmployeePayHistory_BusinessEntityID_RateChangeDate]"
IndexKind="Clustered" />
                  </IndexScan>
                </RelOp>
              </TopSort>
            </RelOp>
          </QueryPlan>
        </StmtSimple>
      </Statements>
    </Batch>
  </BatchSequence>
</ShowPlanXML>')
```

770

解析

使用USE PLAN可以捕捉XML格式的查询执行计划，然后强制查询在随后的执行中使用它。在这个技巧中，使用SET STATISTICS XML ON来捕获查询的XML执行计划定义。然后把这个定义复制到OPTION子句中。USE PLAN提示需要Unicode格式。因此XML文档文本是以N'为前缀的。

USE PLAN和计划指南都应该是不得已的情况下才使用的——在彻底研究过其他可能性之后，比如查询设计、索引、数据库设计、索引碎片和过期的统计信息。USE PLAN可能暂时有效，但随着数据的更新，查询执行计划也需要更新。最后，随着时间的推移，SQL Server也可能会更好地动态决定比你的选择更合适的SQL计划。不过，微软为SQL Server不能选择出足够好的查询执行计划的情况下提供了高级的故障排除选项。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

28.5.3　不修改应用程序的SQL去应用提示

正如在本章开始时提到的，发现并解决查询性能问题需要包括很多方面，例如数据库设计、索引的创建、查询结构。可以修改你的代码，但是如果代码你不能修改呢？

如果你遇到不是你自己的代码而不能修改的数据库或查询问题（例如，封装软件），那样的话你的选择就非常受限制了。通常在这个第三方软件的情况下，你被限制不能去增加新索引或从大数据量的表中归档数据。通常修改厂商的实际数据库对象或查询也是被禁止的。

SQL Server通过使用计划指南（plan guide）为这个普遍的问题提供了解决方案。使用计划指南可以不去修改应用程序中实际的查询文本，而只是应用提示到查询中。

提示　在SQL Server 2008中，可以在计划指南中同时指定查询和表提示。

计划指南可以应用到数据库对象（存储过程、函数、触发器）中指定的查询上，或是指定的独立SQL语句中。

通过使用系统存储过程sp_create_plan_guide创建计划指南：

```
sp_create_plan_guide [ @name = ] N'plan_guide_name'
  , [ @stmt = ] N'statement_text'
  , [ @type = ] N' { OBJECT | SQL | TEMPLATE }'
  , [ @module_or_batch = ]
      {
        N'[ schema_name.]object_name'
        | N'batch_text'
        | NULL
      }
  , [ @params = ] { N'@parameter_name data_type [,...n ]' | NULL }
  , [ @hints = ] { N'OPTION ( query_hint [,...n ] ) ' | N'XML_showplan' | NULL }
```

表28-7描述了这个命令的参数。

表28-7　sp_create_plan_guide参数

参　数	描　述
plan_guide_name	这个参数定义新计划指南的名称
statement_text	这个参数指定了指定要进行调优的SQL文本
OBJECT \| SQL \| TEMPLATE	当选择了OBJECT，计划指南将应用到指定的存储过程、函数、或DML触发器中找到的语句文本。当选择了SQL，计划指南将应用到在独立的语句或批处理中找到的语句文本。TEMPLATE选项决定是否启用SQL语句的参数化。回顾一下22.2.8节的PARAMETERIZATION选项，当将它设置为FORCED，会增加查询被参数化的可能性，允许它形成可重用的查询执行计划。但是，当设为SIMPLE参数化时，仅影响较少数量的查询（依据SQL Server的判断）。如果数据库使用了SIMPLE参数化，你可以强制指定的查询语句为参数化，如果数据库使用了FORCED参数化，你可以强制指定的查询语句为非参数化
N'[schema_name.]object_ name' \| N'batch_text' \| NULL	当选择了TEMPLATE时，这个参数指定SQL文本所在的对象的名称，或批处理文本，再或者是NULL
N'@parameter_name data_type [,...n]' \| NULL	这个参数定义在SQL或TEMPLATE类型的计划指南中使用的参数的名称
N'OPTION (query_hint [,...n])' \| N'XML_showplan' \| NULL	这个参数定义应用到语句的查询提示、要应用的XML查询计划、或者是NULL，它用来表示查询的OPTION子句不生效

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注解 在SQL Server 2008中，现在@hits参数接受将XML Showplan输出作为直接的输入。

使用系统存储过程sp_control_plan_guide删除或禁用计划指南：

```
sp_control_plan_guide [ @operation = ] N'<control_option>'
  [ , [ @name = ] N'plan_guide_name' ]

<control_option>::=
{
    DROP
  | DROP ALL
  | DISABLE
  | DISABLE ALL
  | ENABLE
  | ENABLE ALL
}
```

表28-8描述了这个命令的参数。

772

表28-8 sp_control_plan_guide参数

参 数	描 述
DROP	DROP操作从数据库中删除计划指南
DROP ALL	DROP ALL从数据库中删除所有的计划指南
DISABLE	DISABLE选项禁用计划指南，但是不将它从数据库中删除
DISABLE ALL	DISABLE ALL禁用了数据库中所有的计划指南
ENABLE	ENABLE启用了一个禁用的计划指南
ENABLE ALL	ENABLE ALL启用数据库中所有禁用的计划指南
plan_guide_name	plan_quide_name定义了对其执行操作的计划指南的名称

在这个技巧的示例中，计划指南被用来改变独立查询的表联结类型方法。在这个情况下，第三方软件包发出一个引发LOOP联结的查询。在这个情况下，你希望使用MERGE联结的查询去取代它。

警告 在绝大多数情况下，应该让SQL Server自行做出关于如何处理查询的决策。只有在特定的情形下，并且由富有经验的SQL Server专业人员决定是否在SQL Server环境下创建计划指南。

在这个示例中，执行下面使用了sp_executesql的查询：

```
EXEC sp_executeSQL
N'SELECT v.Name ,a.City
FROM Purchasing.Vendor v
INNER JOIN [Person].BusinessEntityAddress bea
    ON bea.BusinessEntityID = v.BusinessEntityID
INNER JOIN Person.Address a
    ON a.AddressID = bea.AddressID'
```

请看使用SET STATISTICS XML ON显示这个查询执行计划的片断，它显示了使用嵌套循环运算符将表Vendor和BusinessEntityAddress联结到一起：

```
<RelOp NodeId="0" PhysicalOp="Nested Loops" LogicalOp="Inner Join"
EstimateRows="105.447" EstimateIO="0" EstimateCPU="0.000440767"
AvgRowSize="93" EstimatedTotalSubtreeCost="0.322517"
Parallel="0" EstimateRebinds="0" EstimateRewinds="0">
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

例如，如果希望SQL Server使用一个不同的联结方式，但不修改应用程序中实际的查询，则可以通过创建计划指南应用这个修改。随后创建了在应用程序中的查询上应用联结提示的计划指南：

```
EXEC sp_create_plan_guide
   @name = N'Vendor_Query_Loop_to_Merge',
   @stmt = N'SELECT v.Name ,a.City
FROM Purchasing.Vendor v
INNER JOIN [Person].BusinessEntityAddress bea
     ON bea.BusinessEntityID = v.BusinessEntityID
INNER JOIN Person.Address a
     ON a.AddressID = bea.AddressID',
@type = N'SQL',
@module_or_batch = NULL,
@params = NULL,
@hints = N'OPTION (MERGE JOIN)'
```

773

提示　在SQL Server 2008中，也可以在计划指南的@hints参数中指定表提示。

可以通过查询sys.plan_guides目录视图确认计划指南是否已创建（也确认一下设置）：

```
SELECT name, is_disabled, scope_type_desc, hints
FROM sys.plan_guides
```

这个查询返回：

```
name                          is_disabled     scope_type_desc     hints
Vendor_Query_Loop_to_Merge    0               SQL                 OPTION (MERGE JOIN)
```

在创建了计划指南之后，再次使用sp_executesql执行查询。请看XML执行计划，可以看到嵌套循环联结被更改为MERGE联结运算符——并没有修改SQL Server上应用程序中的实际查询：

```
<RelOp NodeId="0" PhysicalOp="Merge Join" LogicalOp="Inner Join"
EstimateRows="105.447" EstimateIO="0" EstimateCPU="0.0470214"
AvgRowSize="93" EstimatedTotalSubtreeCost="0.491476" Parallel="0"
EstimateRebinds="0" EstimateRewinds="0">
```

事实上，查询中所有的联结都从LOOP转化为MERGE联结，这对于多联结查询可能不是预想的效果！如果确定MERGE联结不如嵌套LOOP联结效率更高，可使用sp_control_plan_guide系统存储过程删除计划指南：

```
EXEC sp_control_plan_guide N'DROP', N'Vendor_Query_Loop_to_Merge'
```

解析

计划指南允许不用修改应用程序本身，而增加查询提示到应用程序的查询。在这个示例中，一个特定的SQL语句执行了嵌套LOOP联结。不去修改实际的查询本身，SQL Server“看到”计划指南并将进入的查询匹配到计划指南中的查询。当匹配时，就将计划指南中的提示应用到进入的查询中。

使用sp_create_plan_guide可以为独立的SQL语句、对象（存储过程、函数、DML触发器）中的SQL语句以及不管是否被参数化（取决于数据库的PARAMETERIZATION设置）的SQL语句创建计划。

在这个技巧中，sp_create_plan_guide的第一个参数是新计划指南的名称：

```
EXEC sp_create_plan_guide
    @name = N'Vendor_Query_Loop_to_Merge',
```

774

第二个参数是应用计划指南的SQL语句（忽略空白的字符、注释和分号）：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    @stmt = N'SELECT v.Name ,a.City
FROM Purchasing.Vendor v
INNER JOIN [Person].BusinessEntityAddress bea
    ON bea.BusinessEntityID = v.BusinessEntityID
INNER JOIN Person.Address a
    ON a.AddressID = bea.AddressID',
@type = N'SQL',
```

第三个参数是计划指南的类型，在这个示例中是独立的SQL：

```
@type = N'SQL',
```

对于第四个参数，在这里不是存储过程、函数或触发器，@module_or_batch参数为NULL：

```
@module_or_batch = NULL,
```

@params参数也被设为NULL，因为这不是一个TEMPLATE计划指南：

```
@params = NULL,
```

最后的参数包含应用到传入查询上的实际提示——在这个例子中，强制查询中的所有联结使用MERGE操作：

```
@hints = N'OPTION (MERGE JOIN)'
```

最后，使用系统存储过程sp_control_plan_guide从数据库中删除计划指南，为第一个参数指定DROP操作，然后为第二个参数指定计划指南名称。

28.5.4 从缓存创建计划指南

SQL Server 2008引入使用sp_create_plan_guide_from_handle系统存储过程基于查询计划缓存中既有的查询计划创建查询计划的功能。

在如下的情况下考虑使用这个功能。

- 你需要在升级或数据库迁移后保持查询计划稳定。
- 你有特定的查询使用“不良”计划，你希望它使用已知的“优良”计划。
- 你的应用程序有关于指定响应时间的服务级参数的关键任务查询，并且你希望保持时间稳定。
- 你需要在另一SQL Server实例中重新产生准确的查询执行计划（例如，测试、QA）。
- 你有需要周期性执行的查询，但不必总是以最优化的执行。

警告 应该在绝大多数情况下视需要让SQL Server编译和重编译计划，而不是依赖计划指南。SQL Server能在必要时重编译既有的计划，以适应在查询中的数据分布和对象引用中的任何新变化。 775

sp_create_plan_guide_from_handle系统存储过程的语法如下：

```
sp_create_plan_guide_from_handle [ @name = ] N'plan_guide_name'
    , [ @plan_handle = ] plan_handle
    , [ [ @statement_start_offset = ] { statement_start_offset | NULL } ]
```

表28-9描述了这个命令的参数。

表28-9 sp_create_plan_guide_from_handle的参数

参 数	描 述
plan_guide_name	这个参数定义了新计划指南的名称
plan_handle	这个参数指定了从sys.dm_exec_query_stats动态管理视图中的计划句柄
statement_start_offset \| NULL	语句开始偏移量指定查询批处理中的开始位置。如果为NULL，为批处理中每个语句的查询计划创建计划指南

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使用这个功能可以为将来在SQL Server实例上的重用保留查询计划。在这个技巧中，将演示在缓存中为随后的查询（为了从缓存中取得计划，我先执行它）创建计划指南：

```
SELECT
     p.Title,
     p.FirstName,
     p.MiddleName,
     p.LastName
FROM HumanResources.Employee e
INNER JOIN Person.Person p
     ON p.BusinessEntityID = e.BusinessEntityID
WHERE Title = 'Ms.'
```

执行查询之后，可以通过执行如下的查询来检索指向缓存中查询计划的计划句柄：

```
SELECT plan_handle
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(plan_handle) t
WHERE t.text LIKE 'SELECT%p.Title%' AND
      t.text LIKE '%Ms%'
```

这个查询返回：

```
plan_handle
0x060005006E48752FB080940B000000000000000000000000
```

下一步，使用sp_create_plan_guide_from_handle系统存储过程创建基于计划句柄的计划指南：

```
EXEC sp_create_plan_guide_from_handle 'PlanGuide_EmployeeContact',
@plan_handle = 0x060005006E48752FB080940B000000000000000000000000,
@statement_start_offset = NULL
```

查询sys.plan_handles系统目录视图，可以确认计划指南是否已正确创建（由于查询计划和文本在书中的显示问题，结果未显示）：

776

```
SELECT name, query_text, hints
FROM sys.plan_guides
```

sys.plan_guides中的hits列包含XML格式的查询执行计划。

提示　可以通过跟踪SQL Server Profiler事件“Plan Guide Successful”和“Plan Guide Unsuccessful”来确定计划指南是否被成功使用。

解析

这个技巧演示了如何将既有的缓存中的计划作为计划指南。这是用在匹配计划指南的查询文本的查询上的执行计划。就算在SQL Server实例重启、或刷新过程缓存之后，仍将使用相应的计划指南的查询计划。

首先执行SELECT查询，执行后查询计划就缓存在SQL Server实例中。在完成这个操作之后，可以通过查询sys.dm_exec_query_stats搜索缓存计划的计划句柄。我还对sys.dm_exec_sql_text使用CROSS APPLY，这样就可以搜索到包含在查询中全部的文本。

在获得计划句柄之后，执行sp_create_plan_guide_from_handle系统存储过程。第一个参数是计划指南的名称：

```
EXEC sp_create_plan_guide_from_handle 'PlanGuide_EmployeeContact',
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第二个参数包含计划句柄（注意，应该将计划句柄换成局部变量，然后在一个批处理中用sys.dm_exec_query_stats查询将它放入存储过程中）。

最后，指定语句开始偏移量为NULL。这是因为缓存的计划只包含单个语句。如果这是多语句的批处理，应该为这个参数指定语句开始偏移量数值：

```
@statement_start_offset = NULL
```

在创建计划指南之后，所有匹配的执行SQL将使用在计划指南中指定的查询执行计划（查看sys.plan_guides系统目录视图中的hits列来确认）。这样可以在几种情况下保持计划稳定——例如，在数据库迁移到新的SQL Server实例后、补丁升级之后、或版本升级之后。高度易变的查询执行计划（由于多变的执行计划性能的影响经常重编译）可以从“冻结”到更有效率或关联的查询的执行计划来获得益处。

28.5.5 检查计划指南的有效性

SQL Server 2008引入新的sys.fn_validate_plan_guide系统函数，使用它可以检查既有计划指南的有效性。SQL Server在基于查询中引用对象的变化来编译和重编译查询执行计划方面做了非常棒的工作。从另一方面来讲，计划指南不会根据变化环境自动更改。

sys.fn_validate_plan_guide是只有一个参数的表值函数，这个参数是plan_guide_id。在这个技巧中，演示验证所有在数据库上下文中我感兴趣的计划指南（例如，AdventureWorks数据库中）：

777

```
SELECT pg.plan_guide_id, pg.name, v.msgnum,
       v.severity, v.state, v.message
FROM sys.plan_guides pg
CROSS APPLY sys.fn_validate_plan_guide(pg.plan_guide_id) v
```

如果这个查询没有行返回，意味着既有的计划指南不存在错误。如果返回了行，你就要根据变化环境重建有效的计划指南。

解析

这个技巧演示了如何检查指定数据库中所有计划指南的有效性。SELECT语句引用了计划指南的ID、名称，以及消息编号、严重级别、状态和存在错误的消息：

```
SELECT pg.plan_guide_id, pg.name, v.msgnum,
       v.severity, v.state, v.message
```

FROM子句包含sys.plan_guides，它返回数据库上下文中的所有计划指南：

```
FROM sys.plan_guides pg
```

由于这是需要输入参数的表值函数，我对sys.fn_validate_plan_guide使用CROSS APPLY，然后使用从sys.plan_guides而来的计划指南作为输入：

```
CROSS APPLY sys.fn_validate_plan_guide(pg.plan_guide_id) v
```

这个查询返回由于底层对象的变化而改变的无效计划指南的行。

28.5.6 使用计划指南将非参数化查询参数化

当评估整个SQL Server实例性能的时候，要使用sys.dm_exec_cached_plans动态管理视图来查看SQL Server实例中缓存了什么样的计划。我对objtype列特别感兴趣，查看SQL Server实例的应用程序是否大部分使用了已预定义的语句、存储过程、或即席查询。

对于重度使用即席查询的应用程序，经常看到充满了几乎相同查询的非常大的查询缓存。例如，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如下的查询显示了对象类型和关联的查询文本：

```
SELECT cp.objtype, st.text
FROM sys.dm_exec_cached_plans cp
CROSS APPLY sys.dm_exec_sql_text(cp.plan_handle) st
WHERE st.text LIKE 'SELECT BusinessEntityID%'
GO
```

在我的数据库中，看到返回了3行：

```
objtype     text
Adhoc       SELECT BusinessEntityID
            FROMHumanResources.Employee
            WHERE NationalIDNumber = 509647174

Adhoc       SELECT BusinessEntityID
            FROM HumanResources.Employee
            WHERE NationalIDNumber = 245797967

Adhoc       SELECT BusinessEntityID
            FROM HumanResources.Employee
            WHERE NationalIDNumber = 295847284
```

778

注意，每行都非常相似，除了NationalIDNumber值不同。理想情况下，这种形式的查询应该封装在存储过程中或使用sp_executesql来调用，以防止缓存同一计划并促进计划重用。

如果不能控制执行调用查询的形式，一种做法是使用计划指南强制查询参数化，在这个技巧中将演示它。

在28.5.3节的技巧中，介绍了sp_create_plan_guide系统存储过程。这个过程的TEMPLATE选项用来覆盖数据库的SIMPLE或FORCED参数化选项。如果数据库使用SIMPLE参数化，可以强制指定的查询语句参数化。如果数据库使用FORCED参数化，可以强制指定查询语句不去参数化。

sp_get_query_template系统存储过程通过用sp_create_plan_guide获取查询和它参数化的形式，让使用模板计划指南更加容易。这个过程的语法如下：

```
sp_get_query_template
   [ @querytext = ] N'query_text'
      , @templatetext OUTPUT
      , @parameters OUTPUT
```

表28-10描述了这个命令的参数。

表28-10　sp_get_query_template的参数

参　数	描　述
querytext	希望参数化的查询
templatetext	包含查询参数化形式的输出参数
parameters	包含参数名称和数据类型列表的输出参数

在这个技巧中，先使用sp_get_query_template填充模板SQL和参数，然后将这些值传递到sp_create_plan_guide（我会在“解析”部分一点点地讲解代码）：

```
DECLARE @sql nvarchar(max)
DECLARE @parms nvarchar(max)

EXEC sp_get_query_template
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    N'SELECT BusinessEntityID
FROM HumanResources.Employee
WHERE NationalIDNumber = 295847284',
    @sql OUTPUT,
    @parms OUTPUT
EXEC sp_create_plan_guide N'PG_Employee_Contact_Query',
    @sql,
    N'TEMPLATE',
    NULL,
    @parms,
    N'OPTION(PARAMETERIZATION FORCED)'
```

779

在创建计划指南之后，可以执行同一查询的3个不同版本（3个不同的NationalIDNumber值，每个执行都是独立的且不属于同一批处理）：

```
SELECT BusinessEntityID
FROM HumanResources.Employee
WHERE NationalIDNumber = 295847284

SELECT BusinessEntityID
FROM HumanResources.Employee
WHERE NationalIDNumber = 245797967

SELECT BusinessEntityID
FROM HumanResources.Employee
WHERE NationalIDNumber = 509647174
```

在执行这些查询之后，现在将检查缓存查看这个查询的预定义计划是否存在：

```
SELECT usecounts,objtype,text
FROM sys.dm_exec_cached_plans cp
CROSS APPLY sys.dm_exec_sql_text(cp.plan_handle) st
WHERE st.text LIKE '%(@0 int)SELECT BusinessEntityID%' AND
   objtype = 'Prepared'
```

这个查询返回预定义计划使用过的次数（在计划指南创建之后使用了3次）、对象类型以及参数化的SQL文本：

```
usecounts   objtype     text
3           Prepared    (@0 int)select BusinessEntityID from HumanResources . Employee
                        where NationalIDNumber = @0
```

解析

在这个技巧中，演示了如何强制单个查询参数化。就像我演示的几乎相同的查询不必扩充缓存，消耗内存并产生过度的编译操作。通过减少编译并促进预定义计划的使用，可以提高查询本身的性能并减少在SQL Server实例中的资源消耗。

一点点地看代码，先定义两个用来保存模板SQL和相关参数的局部变量：

```
DECLARE @sql nvarchar(max)
DECLARE @parms nvarchar(max)
```

然后执行对sp_get_query_template系统存储过程的调用：

```
EXEC sp_get_query_template
```

28

这个存储过程的第一个参数希望转化为模板格式的SQL：

780

```
N'SELECT BusinessEntityID
FROM HumanResources.Employee
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
WHERE NationalIDNumber = 295847284',
```

第二个参数是用作包含模板SQL的输出参数：

```
@sql OUTPUT,
```

第三个参数是用作包含用于模板SQL相关联的参数的输出参数：

```
@parms OUTPUT
```

下一步，调用sp_create_plan_guide创建计划指南：

```
EXEC sp_create_plan_guide
```

这个过程的第一个参数是新计划指南的名称：

```
N'PG_Employee_Contact_Query',
```

第二个参数是模板SQL的值：

```
@sql,
```

第三个参数指定它应该是TEMPLATE计划指南：

```
N'TEMPLATE',
```

@module_or_batch参数指定为NULL值，这是TEMPLATE计划指南需要的值：

```
NULL,
```

下一个参数包含所有模板SQL关联的参数定义：

```
@parms,
```

最后一个参数指定了附加到查询的提示。在这个例子中，要求查询使用强制参数化：

```
N'OPTION(PARAMETERIZATION FORCED)'
```

在计划指南创建之后，以3种形式执行查询，每一个都用不同的NationalIDNumber文本值。然后检查sys.dm_exec_cached_plans查看是否存在预处理计划的新行。确认了usecounts列的值为3（其中一个查询执行是我刚执行的），这帮助我确认最新的参数化预定义计划是否被重用。

28.5.7 限制查询资源消耗竞争

SQL Server 2008引入了使用资源调控器限制工作负荷的资源消耗。使用资源调控器可以定义限制CPU任务调度带宽和预留内存的最小值与最大值的资源池。

781 **提示** 当发生跨所有可用调度程序的CPU争用只限制CPU任务调度。

SQL Server提供两个开箱即用的资源池：默认和内部。内部资源池不能修改，它为SQL Server正在处理的活动使用不受限制的资源。默认资源池用在连接和配置资源调控器之前的请求，并且默认情况下在资源上没有限制（尽管之后你可以修改它）。

可以使用CREATE RESOURCE POOL命令创建你自己的资源池。这个命令的语法如下：

```
CREATE RESOURCE POOL pool_name
[ WITH
          ( [ MIN_CPU_PERCENT = value ]
      [ [ , ] MAX_CPU_PERCENT = value ]
      [ [ , ] MIN_MEMORY_PERCENT = value ]
      [ [ , ] MAX_MEMORY_PERCENT = value ] )]
```

表28-11描述了这个命令的参数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表28-11 CREATE RESOURCE POOL的参数

参 数	描 述
Pool_name	这个参数定义资源池的名称
MIN_CPU_PERCENT = value	当发生查询争用，这个参数定义最小有保障的平均CPU任务调度百分比，范围从0到100
MAX_CPU_PERCENT = value	当发生查询争用，这个参数定义所有查询在资源池中请求的最大CPU任务调度百分比
MIN_MEMORY_PERCENT = value	这个参数指定资源池预留内存的最小百分比
MAX_MEMORY_PERCENT = value	这个参数指定资源池中可以用于查询请求的服务器内存最大百分比

一旦创建一个或多个资源池，之后可以将它们关联到工作负荷组。一个或多个工作组可以绑定到一个资源池上。使用工作负荷组可以定义资源池中重要的请求、最大授予内存百分比、最大以秒为单位的CPU时间、最大授予内存超时时间、最大并行度和同时执行的请求的最大数量。可以使用CREATE WORKLAD GROUP创建资源池。这个命令的语法如下所示：

```
CREATE WORKLOAD GROUP group_name
[ WITH
    ( [ IMPORTANCE = { LOW | MEDIUM | HIGH } ]
          [ [ , ] REQUEST_MAX_MEMORY_GRANT_PERCENT = value ]
          [ [ , ] REQUEST_MAX_CPU_TIME_SEC = value ]
          [ [ , ] REQUEST_MEMORY_GRANT_TIMEOUT_SEC = value ]
          [ [ , ] MAX_DOP = value ]
          [ [ , ] GROUP_MAX_REQUESTS = value ] )]
[ USING { pool_name | "default" } ]
```

表28-12描述了这个命令的参数。 782

表28-12 CREATE WORKLOAD GROUP的参数

参 数	描 述
group_name	定义工作负荷组的名称
IMPORTANCE = {LOW \| MEDIUM \| HIGH}	定义工作负荷组中请求的重要性。如果两个工作负荷共享同一资源池，每个工作负荷的重要性决定请求是否需要更高的优先级
REQUEST_MAX_MEMORY_GRANT_PERCENT = value	指定请求可使用资源池中最大内存量
REQUEST_MAX_CPU_TIME_SEC = value	指定以秒为单位的请求可从资源池中使用的最大CPU时间
REQUEST_MEMORY_GRANT_TIMEOUT_SEC = value	指定在失败之前请求可等待的最大秒数
MAX_DOP = value	定义在工作负荷组中请求允许的最大并行度
GROUP_MAX_REQUESTS = value	指定工作负荷组中最大同时执行的请求
USING { pool_name \| "default" }	指定工作负荷组将绑定到哪一个资源池

提示 多个工作负荷组可以关联到一个资源池，但一个工作负荷组不能关联到多个资源池。

正像有内部和默认资源池，也有内部和默认工作负荷组。默认工作负荷组用在没有被分类器用户定义函数（确定哪个工作负荷组接收关联的连接的函数——稍后在这个技巧中演示）覆盖的任何请求上。

在创建用户定义工作负荷组和绑定到资源池后，可以创建一个帮助确定进入的SQL Server连接和请求属于哪个工作负荷组的分类器用户定义函数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

例如，如果有个名为Sue的SQL登录名，可以将分类器函数中的登录名指派到关联到特定资源池的指定工作负荷组。

分类器用户定义函数创建在master数据库中，并且返回进入的SQL Server连接将使用的工作负荷组名称。使用ALTER RESOURCE GOVERNOR命令为进入的连接激活分类器——稍后我将在这个技巧中演示。

在这个技巧中，首先假定SQL Server实例被有两种常规类型活动的应用程序使用。第一种类型的活动与应用程序本身相关。应用程序使用带有指定连接特点并必须可靠运行的持续的自动进程。需要关于事务活动周期性信息（但获得信息时一定不能影响主应用程序的性能）的用户。要承认，最佳实践应该是将这个活动分离到两个SQL Server实例中，但如果这是不可能的，就可以使用资源调控器来限制资源。

先为SQL Server实例创建两个分离的用户定义资源池。第一个资源池将用在高优先级应用程序中。
783 我会确认这个资源池在查询争用时至少保留25%的CPU和内存：

```
CREATE RESOURCE POOL priority_app_queries
WITH ( MIN_CPU_PERCENT = 25,
       MAX_CPU_PERCENT = 75,
       MIN_MEMORY_PERCENT = 25,
       MAX_MEMORY_PERCENT = 75)
GO
```

下一步，将创建为即席查询预留的第二个资源池。将指定这些资源池在高查询争用时的最大CPU和内存为25%，为前面创建的资源池保留资源：

```
CREATE RESOURCE POOL ad_hoc_queries
WITH ( MIN_CPU_PERCENT = 5,
       MAX_CPU_PERCENT = 25,
       MIN_MEMORY_PERCENT = 5,
       MAX_MEMORY_PERCENT = 25)
GO
```

可以使用ALTER RESOURCE POOL命令修改资源池的值。例如，现在继续将即席查询资源池的最小内存修改为10%，最大内存修改为50%：

```
ALTER RESOURCE POOL ad_hoc_queries
WITH ( MIN_MEMORY_PERCENT = 10,
       MAX_MEMORY_PERCENT = 50)
GO
```

在创建完资源池之后，现在可以使用sys.resource_governor_resource_pools目录视图来确认设置：

```
SELECT pool_id,name,min_cpu_percent,max_cpu_percent,
       min_memory_percent,max_memory_percent
FROM sys.resource_governor_resource_pools
```

这个查询返回：

pool_id	name	min_cpu_percent	max_cpu_percent	min_memory_percent	max_memory_percent
1	internal	0	100	0	100
2	default	0	100	0	100
258	ad_hoc_queries	5	25	10	50
259	priority_app_queries	25	75	25	75

现在已经创建了资源池，我可以将工作负荷组绑定到资源池。在这个例子中，先为我最高优先级

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的连接创建工作负荷组。将这个工作负荷组设置为高重要性，并慷慨地授予最高内存百分比和其他参数：

```
CREATE WORKLOAD GROUP application_alpha
WITH
   ( IMPORTANCE = HIGH,
     REQUEST_MAX_MEMORY_GRANT_PERCENT = 75,
     REQUEST_MAX_CPU_TIME_SEC = 75,
     REQUEST_MEMORY_GRANT_TIMEOUT_SEC = 120,
     MAX_DOP = 8,
     GROUP_MAX_REQUESTS = 8 )
USING priority_app_queries
GO
```

784

下一步，将创建另一个与application_alpha共享相同资源池的工作负荷组，但为低重要性和较少的资源消耗能力：

```
CREATE WORKLOAD GROUP application_beta
WITH
    ( IMPORTANCE = LOW,
      REQUEST_MAX_MEMORY_GRANT_PERCENT = 50,
      REQUEST_MAX_CPU_TIME_SEC = 50,
      REQUEST_MEMORY_GRANT_TIMEOUT_SEC = 360,
      MAX_DOP = 1,
      GROUP_MAX_REQUESTS = 4 )
USING priority_app_queries
GO
```

使用ALTER WORLOAD GROUP修改工作负荷组的各种限制，例如：

```
ALTER WORKLOAD GROUP application_beta
WITH ( IMPORTANCE = MEDIUM)
```

前面的两个工作负荷组将共享相同的资源池。现在我再创建一个将绑定到在前面创建的即席资源池的工作负荷组。这个工作负荷组将可以使用即席资源池可用的最大内存：

```
CREATE WORKLOAD GROUP adhoc_users
WITH
    ( IMPORTANCE = LOW,
      REQUEST_MAX_MEMORY_GRANT_PERCENT = 100,
      REQUEST_MAX_CPU_TIME_SEC = 120,
      REQUEST_MEMORY_GRANT_TIMEOUT_SEC = 360,
      MAX_DOP = 1,
      GROUP_MAX_REQUESTS = 5 )
USING ad_hoc_queries
GO
```

完成它之后，通过查询sys.resource_governor_workload_groups目录视图来确认工作负荷组的配置：

```
SELECT name,
       Importance impt,
       request_max_memory_grant_percent max_m_g,
       request_max_cpu_time_sec max_cpu_sec,
       request_memory_grant_timeout_sec m_g_to,
       max_dop,
       group_max_requests max_req,
       pool_id
FROM sys.resource_governor_workload_groups
```

这个查询返回：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

name	impt	max_m_g	max_cpu_sec	m_g_to	max_dop	max_req	pool_id
internal	Medium	25	0	0	0	0	1
default	Medium	25	0	0	0	0	2
application_alpha	High	75	75	120	8	8	256
application_beta	Medium	50	50	360	1	4	256
adhoc_users	Low	100	120	360	1	5	257

785

现在准备创建分类器函数。所有新连接都将调用这个函数。这个函数的逻辑是返回所有连接请求被发送的工作负荷组。分类器函数可以使用几个不同的用在逻辑上的连接相关函数，包括HOST_NAME、APP_NAME、SUSER_NAME、SUSER_SNAME、IS_SRVROLEMEMBER和IS_MEMBER。

警告 确认对这个函数进行适当的调优并快速执行。

创建如下函数，来监视SQL Server登录名和连接主机名称，确认新的连接将分配到哪个工作负荷组：

```
USE master
GO

CREATE FUNCTION dbo.JOEPROD_classifier()
RETURNS sysname
WITH SCHEMABINDING
AS
BEGIN
   DECLARE @resource_group_name sysname

   IF SUSER_SNAME() IN ('AppLogin1', 'AppLogin2')
   SET @resource_group_name = 'application_alpha'

   IF SUSER_SNAME() IN ('AppLogin3', 'AppLogin4')
   SET @resource_group_name = 'application_beta'

   IF HOST_NAME() IN ('Workstation1234', 'Workstation4235')
   SET @resource_group_name = 'adhoc_users'

   -- If the resource group is still unassigned, use default
   IF @resource_group_name IS NULL
   SET @resource_group_name = 'default'

   RETURN @resource_group_name

END
GO
```

现在已经创建了分类器函数，可以使用ALTER RESOURCE GOVERNOR及CLASSIFIER_FUNCTION来激活它：

```
-- 分配分类器函数
ALTER RESOURCE GOVERNOR
WITH (CLASSIFIER_FUNCTION = dbo.JOEPROD_classifier)
GO
```

为了启用配置，执行带有RECONFIGURE选项的ALTER RESOURCE GOVERNOR语句：

```
ALTER RESOURCE GOVERNOR RECONFIGURE
GO
```

使用sys.resource_governor_configuration目录视图验证这个设置：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SELECT OBJECT_NAME(classifier_function_id,DB_ID('master')) Fn_Name,
       is_enabled
FROM sys.resource_governor_configuration
```

786

这个查询返回：

```
Fn_Name                is_enabled
JOEPROD_classifier     1
```

现在进入的新连接的活动将被路由到适当的工作负荷组，并从它们关联的资源池中使用资源。

提示 可以使用sys.dm_resource_governor_resource_pools和sys.dm_resource_governor_workload_groups动态管理视图监视资源池和工作负荷组中进入请求的统计信息。

可以执行ALTER RESOURCE GOVERNOR和DISABLE变量禁用这个设置：

```
ALTER RESOURCE GOVERNOR DISABLE
```

可以通过执行DROP WORKLOAD GROUP和DROP RESOURCE POOL删除用户定义工作负荷组和资源池：

```
USE master
GO

DROP WORKLOAD GROUP application_alpha
DROP WORKLOAD GROUP application_beta
DROP WORKLOAD GROUP adhoc_users

DROP RESOURCE POOL ad_hoc_queries
DROP RESOURCE POOL priority_app_queries
```

当不再分类器函数时可以删除它：

```
ALTER RESOURCE GOVERNOR
WITH (CLASSIFIER_FUNCTION = NULL)

DROP FUNCTION dbo.JOEPROD_classifier
```

解析

这个技巧演示了如何使用资源调控器将内存和CPU资源分配到分离的用户定义资源池中。在定义了资源池之后，创建了工作负荷组，依次为它们指定分配到的用户定义资源池限制的界限。然后创建用来将工作负荷组分配给进入的连接请求的分类器用户定义函数。使用它可以将较低优先级请求限制到比更高优先级请求更少的资源上。

使用这个新功能可以在SQL Server实例上对有各种工作负荷需求和限制的系统资源进行重要控制的维护。即使在有足够的系统资源系统中，还是可以使用资源调控器来保证高优先级的工作负荷不受低优先级请求的负面影响。

787

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 29 章 备份与恢复 29

SQL Server最重要的职责之一就是保护数据。这一章包含了各种备份数据库的技巧，包括完全备份、文件备份、文件组备份、事务日志备份或差异备份（我们会详细地讲解所有这些备份）。

本章还将复习如何使用SQL Server 2008企业版中新增加的压缩改进功能，你也将学习到使用这些备份类型来恢复（还原）数据库的方法。

29.1 创建备份和恢复计划

在详细讲解如何备份和还原SQL Server 数据库之前，我们先来讨论如何生成数据库恢复计划。在通常情况下，你应该思考一下如何回答下面的问题。

- 哪一个数据库比较重要？如果某个数据库是重要的，并且它不只是用来完成用完就丢掉的工作，那么应该备份它。
- 你可以忍受丢失多少数据？换句话说，你的RPO（Recovery Point Objective，恢复点目标）是什么？你能忍受丢失一天的数据吗？或是一小时的？或是一分钟的？你能承受丢失的数据越少，就越要经常进行数据库备份。
- 你是否有异地的存储设施？灾难发生时，设备会被淋湿或在火灾中被烧毁。如果对你来说数据是重要的，你需要通过磁带或网络把它转移到另一个安全的外部地点。
- 你的恢复时间目标（RTO）是什么？你的商业应用可以承受多长的停机时间？在丢失数据库的所有数据后你需要花多少时间才能把所有数据恢复并运行起来？如果你的数据库很大，并且允许的停机时间非常短，你可能需要考虑对你现有的数据库做一个副本（数据库镜像、日志传送、复制或SAN技术）。

恢复计划基于你公司如何看待SQL Server实例及其数据库的价值。对于专用数据库实例来说，其商业价值可能是无关紧要（“随便崩溃”）也可能是关键任务（“不能丢失任何数据”）。不用说大家也都知道关键业务数据库是必须要备份的。如果你不能承受丢失数据或在数据库中重新生成数据，就应该做数据库备份。本章将讨论如何使用T-SQL执行备份操作，以及各种可以执行的备份类型。

备份需要考虑的另一个事项是备份频度。如果你能忍受丢失24小时的数据，那么依据数据库的大小，一天一次的完全数据库备份计划是可以接受的。如果你不能忍受丢失30分钟以上的数据修改，你还是应该考虑每30分钟执行一次事务日志备份。如果你根本不能承受丢失任何数据，那么你应该研究一下像这样的解决方案：日志传送、数据库镜像、RAID镜像或提供存储区域网络（SAN）和分割镜像软件的第三方解决方案。这也暗示了你越要保证没有数据丢失，你就得花越多的钱。

随着备份的进行，也应该考虑将备份生成的文件归档到网络上的另一服务器或磁带上。如果SQL Server实例的主机毁坏了，就肯定需要从离线服务器或异地源上备份了。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

最后一个需要考虑的重点是SQL Server实例和数据库最多允许多少停机时间。先不管丢失的数据，你能承受花多少时间把所有数据恢复并运行起来？每小时的停机时间你的业务会损失多少钱？如果这个数字很高，则需要考虑在冗余上投资来避免停机的影响。如果一个数据库的恢复操作需要8个小时，可能需要重新评价从备份中恢复是否合适，或者说是否划算。在这种情况下，可以选择使用复制、日志传送、数据库镜像或其他在两个或更多的SQL Server实例之间提供有效数据副本的第三方解决方案。故障转移集群也能通过排除单点故障（不包含共享磁盘）帮助提高SQL Server实例的可用性。如果你的硬件损坏了，你在现场有没有可替换的部分？或者是否需要去最近的商店买一个？对于高可用性的需求，需要考虑任何可能发生的单点故障，并且把它们指定为冗余的部分或进程。

作为一名DBA，为了创建SQL Server备份或恢复计划，应该考虑并按照所有在这部分中提到的问题进行操作。退一步讲，你也应该知道在灾难发生的情况下要用到的联系人的详细信息。随后是应该记录的备份及恢复策略的项目清单。

- 你是否有“操作手册”？你需要知道每个连接到数据库的应用程序的主要联系人。谁和最终用户保持联络？如果一个数据库被破坏，谁来做从备份恢复数据（和潜在地丢失一些最近的数据更新）的决定，而不是和微软一起解决被破坏的数据？
- 如果有备用服务器，你的IT部门中谁负责启用备用服务器并运行它？谁来安装操作系统、移动文件、切换DSN名称等？你是否有这些人的列表以及他们的名片 / email / 联系信息？
- 你是否有硬件供应商及软件提供商的支持计划？你是否有列有注册码、服务代码及电话号码的重要文档？
- 你是否有备用部件或可用的备用服务器？
- 如果你的整个现场被破坏了，你是否有另一备选的现场？你是否有移动到这个备选现场的操作手册？

如果整个服务器都被破坏了，并且一定要从头开始重建它，则应该了解更多有用的信息。你的公司应该有如下的记录信息。

- 你的团队中谁负责重建服务器？他们能在凌晨两点进行工作吗？他们是否能在你需要他们的时候出现？
- 你在哪里存放SQL Server的备份文件？你执行的是怎样的备份类型？它们执行的频度如何？
- 是否有其他应用程序安装或配置在SQL Server服务器上？（请记住，除了性能提升，也要保证SQL Server主机为专用的服务器，以减少重新安装第三方或自主开发程序引起的复杂情况。）790
- 数据库运行在什么版本的操作系统之上？你是否有重新安装操作系统需要的光盘？以及重新安装SQL Server的光盘？你是否有所有必需的注册码？
- 你是否记录了安装SQL Server的步骤？要选择何种排序规则？是安装了所有有用的组件（例如Integration Service、Analysis Service、Reporting Service）还是只装了数据库引擎？

要运行在SQL Server实例中的数据库和应用程序越多，就需要准备越多的资料去应对最坏的情况。重要的事情是把它们按重要性排列，首先为你公司最重要的数据库制定计划，然后寻求商业伙伴的帮助，保证你的备份及恢复的计划是更新过且有效的。

29.2 生成备份

在随后的技巧中，我会演示备份SQL Server数据库的各种不同方式。具体来说，我将演示如何执行完全备份、事务日志备份和差异备份。我还将演示SQL Server 2008带来的新压缩改进功能。

完全备份（full backup）会生成一个数据库的副本。当数据库备份操作执行时，数据库对于数据

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

库活动（当数据库处于在线活动状态）来说仍然处于可用状态。在所有的数据库备份选项当中，完全数据库备份是最耗时间的。完全备份包含备份操作完成时刻的所有数据改变和日志文件。一旦生成一个完全数据库备份，它允许你恢复整个数据库。完全备份是数据恢复计划的核心，而且它是利用事务日志或差异备份的先决条件（稍后你将会看到）。当创建备份时，你可以选择在磁盘上创建文件或直接写到磁带上。通常情况下，当SQL Server 备份直接写入磁盘时，备份执行并完成得更快。一旦创建了备份，可以把它复制到磁带或网络驱动器上。

SQL Server数据库需要事务日志文件。事务日志逐条记录了已经提交的事务，及那些已经打开但仍没有提交的事务。这些文件包含正在进行事务的记录以及数据库内的修改。事务日志备份（transaction log backup）备份了最后的完全备份或事务日志备份完成后发生的事务日志的活动。当备份完成后，SQL Server截断日志不活动的部分（这部分不包含打开的事务活动）。事务日志备份具有低资源消耗的特性，并且可以频繁执行（例如，每15分钟执行一次）。

事务日志备份只可以在置为FULL或BULK_LOGGED恢复模式下的数据库上执行。回忆一下第22章提到的3个数据库恢复模式，它们是FULL、BULK_LOGGED和SIMPLE。

- 当置为SIMPLE恢复模式时，SQL Server会自动截断事务日志，取消了该模式执行事务日志备份的能力。在这个恢复模式下，丢失数据的风险依赖于完全或差异备份的时间安排，并且你将不能执行事务日志备份提供的时点恢复。
- BULK_LOGGED恢复模式允许执行完全备份、差异备份和事务日志备份——但是，在大容量操作时，最小限度的日志被记录到事务日志。这种恢复模式的优点是在执行大容量操作时减少了日志空间的使用，作为代价，事务日志备份只能用来恢复到最后事务日志备份完成的时间（不允许时点恢复及标记事务）。
- 在FULL恢复模式下，完整地记录所有事务活动，包括大容量操作。在这个最安全的模式下，所有的还原选项都是可用的，包括时点事务日志还原、差异备份和完全数据库备份。

791

除了允许从完成事务日志备份的完成点上执行还原操作，事务日志备份还允许执行时点恢复以及事务标记恢复。时点恢复对于将数据库还原到数据库被修改或出现故障之前的状态是有用的。事务标记允许你恢复到标记了事务的第一个实例（使用了BEGIN TRAN...WITH MARK）并且包括该事务内所做的更新。

事务日志备份文件的大小取决于数据库活动的级别以及你使用了FULL或BULK_LOGGED恢复模式中的何种。再重申一下，SIMPLE恢复模式是不允许事务日志备份的。

从事务日志备份中恢复，必须先从完全备份中还原，然后再附加事务日志备份。事务日志是积累的，意思是每一个备份都是事务日志备份序列的一部分，而且必须以相同的顺序连续地还原。比如说，你不能在还原完全数据库备份之后就还原第三个事务日志备份，而跳过前两个事务日志备份。

只有在附加完所有你希望按备份的时间顺序附加的事务日志之后，数据库才会被恢复（意思是使其在线并可正常使用）。通过RESTORE命令的RECOVERY和NORECOVERY子句控制恢复，我们会在本章稍后介绍这些内容。

你必须了解已经做过的备份——备份中包含了什么，以及在你能还原它们之前何时执行过备份。在本章稍后我会演示可以用来查看这些信息的各种命令。随后地列表详细地列出了一个典型的备份序列：

时　间	备份类型
上午8点	完全数据库备份
上午10点	事务日志备份
下午1点	事务日志备份

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果希望将数据库恢复到下午1点，首先需要还原上午8点的完全备份，下一步还原上午10点的事务日志备份，最后是下午1点的事务日志备份。如果使用差异备份，必须先还原完全备份，下一步是还原差异备份，然后是在差异备份之后创建的事务日志备份。*差异备份*（differential backup）复制最后一次完全备份之后的所有数据和日志页。由于当数据库备份时它是处于在线状态的，因此差异备份包含从备份开始到备份结束时间点发生的数据改变和日志文件。通过差异备份产生的文件通常要比完全数据库备份的小，并且创建得更快。

差异备份不像事务日志备份，它是自包含的并且只需要还原的数据库最后的完全备份。但是事务日志备份是序列文件，并且不包含前一个事务日志备份的数据。例如，如果你在上午8点运行完全备份，在上午10点进行差异备份，并且在下午1点进行一个补充的差异备份，下午1点的差异备份将仍然包含上午8点的完全备份之后发生的数据修改：

时 间	备份类型
上午8点	完全数据库备份
上午10点	差异备份（捕捉上午8点—上午10点的数据修改）
下午1点	差异备份（捕捉上午8点—下午1点的数据修改）

792

尽管在任何完全备份和差异备份还原之前，事务日志备份不能还原，但是差异备份还是可以与事务日志备份并行工作。

在这组备份技巧的第一个技巧中，会演示如何进行一个最简单的完全备份。

29.2.1 执行基本的完全备份

可以使用BACKUP DATABASE命令执行一个完全备份。执行一个完全备份到磁盘的简化语法如下：

```
BACKUP DATABASE { database_name | @database_name_var }
TO DISK = { 'physical_backup_device_name' | @physical_backup_device_name_var }
[ ,...n ]
```

此命令的参数说明见表29-1。

表29-1 BACKUP DATABASE 参数

参 数	描 述
database_name \| @database_name_var	这个参数定义要备份的数据库名称（可以指定为字符串或局部变量）
'physical_backup_device_name' \| @physical_backup_device_name_var	这个参数指定物理路径和文件名，或包含物理路径和文件名的局部变量
[,...n]	可以为一条BACKUP DATABASE命令指定最多64个备份设备名称

BACKUP命令也包含其他一些选项，大部分选项我们会在本章中演示：

```
[ WITH ] [Option Name ] [,...n]
```

提示 完整的选项列表请看SQL Server 2008联机丛书。我将在本章演示较常用的选项。

在这个技巧中，将执行一个简单的、TestDB数据库的完全数据库备份到一个磁盘设备（文件）上。为了演示BACKUP DATABASE，先创建一个新的临时数据库，其中包含几个来自AdventureWorks数据库的对象：

```
USE master
GO
```

29

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'TestDB')
BEGIN
      CREATE DATABASE TestDB
END
GO

USE TestDB
GO

SELECT *
INTO dbo.SalesOrderDetail
FROM AdventureWorks.Sales.SalesOrderDetail
GO

SELECT *
INTO dbo.SalesOrderHeader
FROM AdventureWorks.Sales.SalesOrderHeader
GO
```

793

下面，备份新的数据库：

```
BACKUP DATABASE TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Oct_14_2008_1617.BAK'
```

这个查询返回：

```
已为数据库'TestDB'，文件'TestDB'(位于文件1上)处理了2456页。
已为数据库'TestDB'，文件'TestDB_log'(位于文件1上)处理了5页。
BACKUP DATABASE成功处理了2461页，花费4.210秒(4.788MB/秒)。
```

解析

在这个简单的技巧中，我们对TestDB数据库进行了完全备份。代码的第一行指定了要备份的数据库名称：

```
BACKUP DATABASE TestDB
```

代码的第二行指定了数据库备份的文件：

```
TO DISK = 'C:\Apress\Recipes\TestDB_Oct_14_2008_1617.BAK'
```

我们创建了一个a*.bak扩展名的备份文件。备份的名称显示了日期及军事时间。尽管文件名中包含的时间戳帮助识别备份创建的时间，但它不是必需的。在执行之后，会返回关于处理过的数据页的数量和备份操作花费的时间信息。

29.2.2 压缩备份

SQL Server 2008在企业版与开发版中引入了原生备份压缩。使用这个功能可以更快速地备份数据库并消耗更少的磁盘空间。压缩量依赖于数据库中的数据。例如，含有重复值字符数据的数据库可以有比包含更多数字或加密数据的数据库更高的压缩率。

在这个技巧中，先演示如何启用SQL Server实例的默认压缩。完成它之后，将展示如何显式不压缩备份或忽略服务器级选项不压缩备份。

在第一个查询中，配置服务器实例的默认值（再次提醒，这需要使用企业版或开发版）：

```
USE master
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
EXEC sp_configure 'backup compression default', '1'
RECONFIGURE WITH OVERRIDE
GO
```

794

执行如下查询，来确认设置是启用的：

```
SELECT description,value_in_use
FROM sys.configurations
WHERE name = 'backup compression default'
```

这个查询返回：

```
description                                value_in_use
Enable compression of backups by default   1
```

现在对AdventureWorks数据库运行简单的备份：

```
BACKUP DATABASE AdventureWorks
TO DISK = 'C:\Apress\AW_compressed.bak'
```

这个操作的统计信息如下：

```
已为数据库'AdventureWorks'，文件'AdventureWorks 2008_Data'（位于文件1 上）处理了24056 页。
已为数据库'AdventureWorks'，文件'FileStreamDocuments'（位于文件1 上）处理了36 页。
已为数据库'AdventureWorks'，文件'AdventureWorks 2008_log'（位于文件1 上）处理了5 页。
BACKUP DATABASE 成功处理了24112 页，花费6.318秒(29.814 MB/秒)。
```

如果不希望压缩备份，可以使用NO_COMPRESSION参数（相同地，如果服务器选项禁用了，也可以指定COMPRESSION来覆盖它）：

```
BACKUP DATABASE AdventureWorks
TO DISK = 'C:\Apress\AW_uncompressed.bak'
WITH NO_COMPRESSION
```

操作统计信息如下所示（注意压缩的备份运行了6.318秒，未压缩的备份用了11.863秒——而且这不是仅有的差别，正如你将会发现的）：

```
已为数据库'AdventureWorks'，文件'AdventureWorks 2008_Data'（位于文件1 上）处理了24056 页。
已为数据库'AdventureWorks'，文件'AW2'（位于文件1 上）处理了16 页。
已为数据库'AdventureWorks'，文件'FileStreamDocuments'（位于文件1 上）处理了36 页。
已为数据库'AdventureWorks'，文件'AdventureWorks 2008_log'（位于文件1 上）处理了1 页。
BACKUP DATABASE 成功处理了24109 页，花费11.863 秒(15.876 MB/秒)。
```

可以通过查看msdb..backupset来查询这两个备份的统计信息：

```
SELECT TOP 2 database_name, backup_size, compressed_backup_size
FROM msdb..backupset
ORDER BY backup_finish_date DESC
```

795

看一下结果中备份占用的空间，AdventureWorks数据库的压缩备份使用了45MB，未压缩的备份使用了191MB（如下数据以字节为单位）：

```
database_name       backup_size     compressed_backup_size
AdventureWorks      200365056       200365056
AdventureWorks      200365056       47361034
```

解析

使用备份压缩可以更快运行并使用更少磁盘空间来创建数据库备份。制衡的消耗是CPU，所以你应该评估对当前查询流量和系统资源需求的负载。

29

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

通过用sp_configure或只是在BACKUP操作的WITH子句中指定COMPRESSION启用backup compression default服务器选项来使用默认压缩。如果希望在服务器选项启用时不压缩指定备份，只要指定NO_COMPRESSION就可以了，正如这个技巧中所演示的。

查询msdb..backupset表的compressed_backup_size列来查看备份压缩的大小。如果数据库备份没有压缩，这个值将与backup_size列的大小相同。

29.2.3　命名和描述备份和媒体

考虑到信息的行业管理和保存法，公司的规定可能需要将数据库备份保留较长一段时间。由于较长的保留周期，备份集元数据变得更加重要。用数据库名和时间戳对数据库备份文件进行命名通常就足够了，可是SQL Server还包含了其他选项，可以利用它们来描述和命名备份。这些选项包括：

```
[ WITH
    [ [ , ] DESCRIPTION = { 'text' | @text_variable } ]
    [ [ , ] MEDIADESCRIPTION = { 'text' | @text_variable } ]
    [ [ , ] MEDIANAME = { media_name | @media_name_variable } ]
    [ [ , ] NAME = { backup_set_name | @backup_set_name_var } ]
]
```

这些选项的描述请见表29-2。

表29-2　备份媒体选项

参　数	描　述
DESCRIPTION	说明备份集的自由格式文本，帮助识别备份设备的内容
MEDIADESCRIPTION	媒体集的自由格式文本说明，帮助识别媒体的内容
MEDIANAME	整个备份媒体集的名称，最多为128个字符
NAME	备份集的名称

796

在前面的表格中提到了两个关于SQL Server备份的术语：备份集和媒体集。简单地说，备份集（backup set）是数据库备份操作的结果。备份集可以跨越一个或多个备份设备（磁盘或磁带）。媒体集（media set）是备份集写入的一个或多个备份设备的集合。

随后的示例演示了为备份集和媒体集指定描述和名称的操作：

```
BACKUP DATABASE TestDB
TO DISK  = 'C:\Apress\Recipes\TestDB.bak'
WITH  DESCRIPTION = 'My second recipe backup, TestDB',
   NAME = 'TestDB Backup October 14th',
   MEDIADESCRIPTION = 'Backups for October 2008, Week 2',
   MEDIANAME = 'TestDB_October_2008_Week2'
```

这个查询返回：

```
已为数据库'TestDB'，文件'TestDB'(位于文件1上)处理了2440页。
已为数据库'TestDB'，文件'TestDB_log'(位于文件1上)处理了1页。
BACKUP DATABASE成功处理了2441页，花费0.815秒(23.399MB/秒)。
```

解析

本技巧演示了如何在数据库备份中增加更多的描述信息。通过WITH子句为BACKUP DATABASE命令增加额外的选项。DESCRIPTION描述了备份集：

```
WITH DESCRIPTION = 'My second recipe backup, TestDB',
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

NAME指定了备份集的名称：

```
NAME = 'TestDB Backup October 14th',
```

MEDIADESCRIPTION指定了媒体集的描述：

```
MEDIADESCRIPTION = 'Backups for October 2008, Week 2',
```

MEDIANAME指定了整个备份媒体集的名称：

```
MEDIANAME = 'TestDB_October_2008_Week2'
```

这个信息可以通过使用RESTORE命令检索（例如RESTORE HEADERONLY），我们会在本章稍后介绍它。

29.2.4 配置备份保持期

在本章的第一个技巧中，如果在备份之前备份文件（设备）不存在，它会在BACKUP命令的执行期间被创建。如果文件已经存在，默认备份过程的行为是在既有的备份文件上附加备份数据（保留文件上的所有其他备份）。

一些BACKUP选项会影响备份集的保持期：

```
[ WITH
    [ [ , ] EXPIREDATE = { date | @date_var }
    | RETAINDAYS = { days | @days_var } ]
    [ [ , ] { FORMAT | NOFORMAT } ]
    [ [ , ] { INIT | NOINIT } ]
    [ [ , ] { NOSKIP | SKIP } ]
]
```

797

这些选项的描述请见表29-3。

表29-3 备份保持期选项

参 数	描 述
EXPIREDATE \| RETAINDAYS	EXPIREDATE指定备份集到期和允许被覆盖的日期。RETAINDAYS指定必须经过多少天才可以覆盖该备份媒体集
FORMAT \| NOFORMAT	FORMAT将在用于此备份操作的所有卷上写入一个新的媒体标头。既有的标头会被覆盖。如果现有设备中存在条带（条带稍后会讲述），则整个媒体集都将变得不可用。NOFORMAT指定不应将媒体标头写入用于此备份操作的所有卷
INIT \| NOINIT	INIT指定应覆盖既有备份集，但是保留媒体标头。如果备份集没有过期，或BACKUP语句中的备份集名与备份媒体上的名称不匹配，备份集将不会被覆盖，NOINIT表示备份集将追加到磁盘或磁带设备上，NOINIT是默认选项
NOSKIP \| SKIP	SKIP禁用备份集的过期和名称检查。NOSKIP检测日期与名称，而且它是确认备份不会被不适当地覆盖的额外的安全措施

这个技巧演示了执行一个备份集保持期设置为30天的完全数据库备份，在这个期限后备份集才可以被覆盖：

```
BACKUP DATABASE TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Oct.bak'
WITH RETAINDAYS = 30
```

现在我们尝试覆盖一个既有的备份，这个备份在TestDB_Oct.bak文件上：

```
BACKUP DATABASE TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Oct.bak'
WITH INIT
```

29

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个查询返回的信息是：

```
消息4030，级别16，状态1，第1行
设备'C:\Apress\Recipes\TestDB_Oct.bak'上的媒体于2008-08-14  2:35:42:000PM过期，无法覆盖。
消息3013，级别16，状态1，第1行
BACKUP DATABASE 正在异常终止。
```

解析

在这个技巧中，创建了一个新的、备份集保持期为30天的数据库备份。在备份创建之后，又执行了另一个备份，这次使用了INIT开关（这个操作将覆盖既有的备份集）。这个尝试失败了，返回了一个关于备份集尚未过期的错误警告，所以备份集不能被覆盖。

798

29.2.5　条带化备份集

条带化（stripping）备份使用一个以上的设备（磁盘或磁带）进行单个备份集操作。事实上，当执行数据库备份时，可以在备份操作中使用最多64个设备（磁盘或磁带）。这个特性在备份非常大的数据库时是非常有用的，因为能通过在分离的磁盘 / 阵列条带化备份文件增强备份性能。条带化备份文件意思是说同时以对称的方式写入每个文件。条带化备份使用并行写操作，可以显著提高备份操作的速度。

此技巧演示了在3个磁盘设备上条带化备份：

```
BACKUP DATABASE TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Stripe1.bak',
   DISK = 'D:\Apress\Recipes\TestDB_Stripe2.bak',
   DISK = 'E:\Apress\Recipes\TestDB_Stripe3.bak'
```

这个备份创建了3个文件，每个文件存储了三分之一还原数据库时需要的备份信息。如果尝试在其中任何一个设备上进行备份，将得到错误消息，正如下一个示例所演示的：

```
BACKUP DATABASE TestDB
TO DISK  = 'C:\Apress\Recipes\TestDB_Stripe1.bak'
```

这个查询返回：

```
消息3132，级别16，状态1，第1行
媒体集有3个媒体簇，但只提供了1个。必须提供所有成员。
消息3013，级别16，状态1，第1行
BACKUP DATABASE正在异常终止。
```

解析

在这个技巧中，备份使用了3个设备，它也被称作媒体簇（media families）。这3个媒体簇被用作单个媒体集，这个媒体集可以包含一个或多个备份集。在创建了由3个媒体簇组成的媒体集之后，技巧的第二部分尝试在一个既有的媒体簇上进行备份操作。这将引发一个错误，因为在文件被格式化（使用WITH FORMAT）之前，必须同时使用它们而不可以在备份操作中单独使用。

29.2.6　使用命名的备份设备

可以在BACKUP或RESTORE命令中定义磁带或磁盘的逻辑名称。定义设备会把它添加到sys.backup_devices目录视图，不用再键入磁盘路径和文件或磁带名称。

为了增加一个新的备份设备定义，可以使用sp_addumpdevice系统存储过程：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
sp_addumpdevice [ @devtype = ] 'device_type'
        , [ @logicalname = ] 'logical_name'
        , [ @physicalname = ] 'physical_name'
    [ , { [ @cntrltype = ] controller_type |
          [ @devstatus = ] 'device_status' }
    ]
```

799

这些命令的参数描述请见表29-4。

表29-4 sp_addumpdevice参数

参 数	描 述
device_type	这个参数用来指定设备类型：disk或tape
logical_name	这个选项定义将在BACKUP和RESTORE中使用的备份设备的名称
physical_name	这个参数定义操作系统文件名（遵守通用命名约定（UNC）的名称）或磁带路径
controller_type	这个参数已忽略（向后兼容）
device_status	这个参数已忽略（向后兼容）

要查看备份设备的定义，可以使用sp_helpdevice系统存储过程，它只包含一个参数logical_name：

```
sp_helpdevice [ [ @devname = ] 'name' ]
```

删除备份设备使用sp_dropdevice：

```
sp_dropdevice [ @logicalname = ] 'device'
    [ , [ @delfile = ] 'delfile' ]
```

第一个参数是备份设备的名称，当第二个参数指定了DELFILE时，将删除实际的备份设备文件。在示例中的第一部分，创建了一个名为TestDBBackup的备份设备，它被映射到C:\Apress\Recipes\TestDB_Device.bak文件上：

```
USE master
GO
```

这个查询返回：

```
命令已成功完成。
```

下一步，使用sp_helpdevice查询关于设备的信息：

```
EXEC sp_helpdevice 'TestDBBackup'
```

800

这个查询返回：

```
device_name     physical_name                          description
TestDBBackup    C:\Apress\Recipes\TestDB_Device.bak     disk, backup device
```

下一步，执行基于该设备的备份：

```
BACKUP DATABASE TestDB
TO TestDBBackup
```

这个查询返回：

```
已为数据库'TestDB'，文件'TestDB'(位于文件1上)处理了2440页。
已为数据库'TestDB'，文件'TestDB_log'(位于文件1上)处理了1页。
BACKUP DATABASE成功处理了2441页，花费0.858秒(22.226MB/秒)。
```

29

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

最后，我们使用sp_dropdevice删除该设备（如果没有指定第二个选项DELFILE，物理备份文件还将保留在操作系统内）：

```
EXEC sp_dropdevice 'TestDBBackup'
```

这个查询返回：

```
设备已删除。
```

解析

在这个技巧中，演示了如何创建一个命名的备份设备，使用它可以不去键入你需要在BACKUP或RESTORE命令中指定的完整磁盘或磁带名称。

在此技巧中的第一个示例使用sp_addumpdevice创建了一个设备。存储过程的第一个参数设置设备类型为disk。第二个参数是设备的逻辑名称，第三个参数是实际的物理文件路径和名称。此技巧的第二个查询演示了使用sp_helpdevice来返回设备的信息。信息包括与设备描述相关的status字段，以及指定设备类型（2代表磁盘设备，5代表磁带）的cntrltype列。此技巧中的第三个查询演示了在备份中使用设备，它只是指定了设备名而不是使用DISK或TAPE选项，在此技巧的最后一个查询中，我们使用sp_dropdevice删除了该设备。

29.2.7　镜像备份集

你可以镜像一个数据库备份、日志备份、文件或文件组备份。镜像通过创建2个、3个或4个媒体集的副本创建备份冗余。因为可以使用其他任何一个可用的镜像媒体集，如果某个媒体集被破坏或不可用，此冗余就可以派上用场了。

语法如下：

```
BACKUP DATABASE { database_name | @database_name_var }
TO < backup_device > [ ,...n ]
[ [ MIRROR TO < backup_device > [ ,...n ] ] [ ...next-mirror ] ]
```

MIRROR TO命令用来连接一个或多个备份设备列表，并且最多支持3个镜像。在这个示例中，一个备份被镜像为3个不同的副本。不像之前的条带化的示例，在数据库恢复操作中实际只需要一个生成的备份文件。但是如果该文件不可用了，可以尝试从其他3个副本中还原：

```
BACKUP DATABASE TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Original.bak'
MIRROR TO DISK = 'D:\Apress\Recipes\TestDB_Mirror_1.bak'
MIRROR TO DISK = 'E:\Apress\Recipes\TestDB_Mirror_2.bak'
MIRROR TO DISK = 'F:\Apress\Recipes\TestDB_Mirror_3.bak'
WITH FORMAT
```

801

这个查询返回：

```
已为数据库'TestDB'，文件'TestDB'(位于文件1上)处理了2456页。
已为数据库'TestDB'，文件'TestDB_log'(位于文件1上)处理了1页。
BACKUP DATABASE成功处理了2457页，花费11.460秒(1.756MB/秒)。
```

这第二个示例演示了镜像一个条带化备份：

```
BACKUP DATABASE TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Stripe_1_Original.bak',
DISK = 'D:\Apress\Recipes\TestDB_Stripe_2_Original.bak'
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
MIRROR TO DISK = 'E:\Apress\Recipes\TestDB_Stripe_1_Mirror_1.bak',
DISK = 'F:\Apress\Recipes\TestDB_Stripe_2_Mirror_1.bak'
WITH FORMAT
```

这个查询返回：

```
已为数据库'TestDB'，文件'TestDB'(位于文件1上)处理了2456页。
已为数据库'TestDB'，文件'TestDB_log'(位于文件1上)处理了1页。
BACKUP DATABASE成功处理了2457页，花费7.883秒(2.553MB/秒)。
```

解析

在此技巧的第一个示例中，将一个备份生成了3个镜像。它为TestDB数据库生成了4个备份文件。第一行代码指定了要备份的数据库：

```
BACKUP DATABASE TestDB
```

第二行代码指定了主（非镜像）备份文件的位置：

```
TO DISK = 'C:\Apress\Recipes\TestDB_Original.bak'
```

后面的3行指定了备份的3个镜像副本：

```
MIRROR TO DISK = 'D:\Apress\Recipes\TestDB_Mirror_1.bak'
MIRROR TO DISK = 'E:\Apress\Recipes\TestDB_Mirror_2.bak'
MIRROR TO DISK = 'F:\Apress\Recipes\TestDB_Mirror_3.bak'
WITH FORMAT
```

注意在第一次创建镜像备份集时，需要使用WITH FORMAT。原始备份放置到了C:\盘下，然后每个镜像的副本分别放置到它们自己所在的盘符（D:\、E:\、F:\）下。在此示例中的任何一个.bak文件都可用来还原TestDB数据库，从而为备份文件损坏的情况提供了冗余。

在此技巧中的第二个示例演示了镜像一个条带化的备份（在一个媒体集中有两个媒体簇）。这次TO DISK包含了用来条带化原始备份的两个文件：

```
BACKUP DATABASE TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Stripe_1_Original.bak',
DISK = 'D:\Apress\Recipes\TestDB_Stripe_2_Original.bak'
```

MIRROR TO DISK也指定了两个文件，它们用来放置原始条带化备份的镜像副本：

```
MIRROR TO DISK = 'E:\Apress\Recipes\TestDB_Stripe_1_Mirror_1.bak',
DISK = 'F:\Apress\Recipes\TestDB_Stripe_2_Mirror_1.bak'
WITH FORMAT
```

注意，MIRROR TO DISK只指定了一次，跟随在它后面的是镜像操作的两个设备。 802

29.2.8 执行事务日志备份

BACKUP LOG命令用来执行事务日志备份。随后是执行事务日志备份的基本语法：

```
BACKUP LOG { database_name | @database_name_var }
{
    TO <backup_device> [ ,...n ]
[ [ MIRROR TO <backup_device> [ ,...n ] ] [ ...next-mirror ] ]
    [ WITH ] [Option Name ] [,...n]
}
```

BACKUP LOG用了许多与BACKUP DATABASE相同的选项和功能。本章还没有演示的选项以及事务日志备份特有的选项在表29-5中进行了描述。

29

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表29-5　BACKUP LOG选项

参　数	描　述
NO_TRUNCATE	如果数据库损坏了，NO_TRUNCATE允许通过不截断不活动的部分（不活动的部分包含已提交的一批事务）来备份事务日志。我们经常在紧急的事务日志备份中用到它，它可以捕捉在RESTORE操作之前的活动事务。不要长期在这种方式下运行，因为日志文件容量将会一直增长
NORECOVERY \| STANDBY =	NORECOVERY备份事务日志的尾部，然后让数据库处于RESTORING状态（它是一个还会有附加RESTORE命令执行的状态）。STANDBY也备份事务日志的尾部，但不会将数据库处于RESTORING状态，而把它置为只读STANDBY状态（用于日志传送）。此选项需要指定一个文件，如果进行了日志还原操作，该文件将保存产生回滚所做的更改

在此技巧的第一个查询中，执行TestDB数据库的事务日志备份：

BACKUP LOG TestDB

```
BACKUP LOG TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Oct_14_2008_1819.trn'
```

这个查询返回：

```
已为数据库'TestDB'，文件'TestDB_log'(位于文件1上)处理了13页。
BACKUP LOG成功处理了13页，花费0.448秒(0.230MB/秒)。
```

此技巧中的第二个例子演示了在事务日志尾部生成一个事务日志备份的操作。假定数据库发生了错误——执行“尾部”的备份意味着备份数据库最新的事务，而没去截断事务日志中不活动的部分：

```
BACKUP LOG TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Oct_14_2008_1820_Emergency.trn'
WITH NO_TRUNCATE
```

803

解析

在此技巧中，演示了两个关于事务日志备份的示例。注意，除非数据库以前进行过完全备份，否则BACKUP LOG不可以执行。而且在两个示例中，数据库必须使用FULL或BULK_LOGGED恢复模式。第一个示例是标准的备份到磁盘的事务日志备份。第一行代码指定了要备份的数据库名称：

```
BACKUP LOG TestDB
```

第二行代码指定了备份操作的设备：

```
TO DISK = 'C:\Apress\Recipes\TestDB_Oct_14_2008_1819.trn'
```

备份完成之后，生成了一个文件，并且事务日志中不活动的部分被自动截断了。在第二个查询中，指定了WITH NO_TRUNCATE选项，使用它可以备份事务日志中活动的部分，而且并不截断事务日志中不活动的部分。

在本章中，稍后你会学习如何从事务日志文件中还原数据，包括如何使用时点恢复。

29.2.9　不破坏备份序列创建备份

数据库和事务日志备份可以使用COPY_ONLY选项来创建备份，这些备份不影响正常的备份序列。你会在本章随后的技巧中看到，差异备份和事务日志备份都依赖于之前执行的完全备份。无论何时创建了其他的完全备份，备份序列都将重新开始。它的意思是之前生成的差异或日志备份不能使用后来生成的完全备份。只有那些在完全备份生成之后创建的差异或事务日志备份才可以使用。

但是，当使用COPY_ONLY选项时，完全备份不会破坏备份序列。这在有重大数据库改变之前需要创建特殊备份时是有用的，此时你不希望破坏标准的备份序列，但可能希望做可以有效RESTORE的“以防

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

万一”的完全备份。这个示例演示如何对完全数据库备份使用COPY_ONLY：

```
BACKUP DATABASE TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Copy.bak'
WITH COPY_ONLY
```

当在事务日志备份中使用COPY_ONLY时，在备份创建之后事务日志不会被截断（保留未破坏的事务日志备份链）。这个示例演示了如何在事务日志备份中使用COPY_ONLY：

```
BACKUP LOG TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Copy.trn'
WITH COPY_ONLY
```

解析

此技巧演示了使用COPY_ONLY创建完全备份和事务日志备份。语法与前面的技巧中的相似，不同之处就是COPY_ONLY包含在WITH子句中。使用了该选项的完全数据库备份不会破坏恢复操作所需的、之前生成的事务日志或差异备份序列。使用了COPY_ONLY的事务日志备份也不会破坏其他事务日志备份的时间顺序。

804

29.2.10 执行差异备份

在这个随后的技巧中，演示了如何创建差异备份。回顾一下本章中前面提到的差异备份，它用来备份在最后一次完全备份之后发生改变的所有数据和日志页。这点与事务日志备份不同，事务日志备份仅捕捉在最后事务日志和/或完全数据库备份之后发生的改变。

差异备份通过使用BACKUP DATABASE来执行，并且使用了与常规完全数据库备份相同的语法和功能——只包括DIFFERENTIAL关键字。此技巧演示了在TestDB数据库上创建一个差异备份：

```
BACKUP DATABASE TestDB
TO DISK = N'C:\Apress\Recipes\TestDB.diff'
WITH DIFFERENTIAL, NOINIT, STATS = 25
```

这个查询返回：

```
已处理百分之58.
已处理百分之78.
已为数据库'TestDB'，文件'TestDB'(位于文件1上)处理了40页.
已处理百分之百.
已为数据库'TestDB'，文件'TestDB_log'(位于文件1上)处理了1页.
BACKUP DATABASE WITH DIFFERENTIAL成功处理了41页，花费0.339秒(0.989MB/秒).
```

解析

在此技巧中，在TestDB数据库上创建了一个差异备份。命令的用法与前面技巧中的相似，只是这次WITH子句中包含了DIFFERENTIAL关键字。还使用了其他两个选项（当使用其他备份类型时也适用）：NOINIT，附加备份集到既有的磁盘或磁带设备上，以及STATS，在备份过程中返回反馈信息到客户端。差异备份在完全数据库备份之后才可执行，所以对于新的数据库，差异备份并不是初始的备份方法。

29.2.11 备份单个文件或文件组

对于超大型数据库来说，如果完全备份的时间超过你的备份时间间隔，可以使用另一个备份指定文件组或文件的选项来减少备份时间。使用这个选项可以在丢失文件或文件组的情况下执行恢复操作。为了在激活读写功能的数据库上执行文件或文件组备份，数据库必须设置为FULL或BULK日志恢复模式，事务日志备份必须在恢复文件或文件组的操作后执行。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

除了需要使用FILEGROUP或FILE关键字及可以指定用逗号分隔的多个文件或文件组之外，备份文件或文件组的操作实际上使用了与完全数据库备份操作相同的语法。

为了演示备份文件组的操作，先创建一个新数据库，它使用一个称为FG2的辅助文件组：

```
USE master
GO

CREATE DATABASE VLTestDB
ON PRIMARY
( NAME = N'VLTestDB',
FILENAME =
N'c:\Apress\Recipes\VLTestDB.mdf' ,
SIZE = 3048KB ,
FILEGROWTH = 1024KB ),
 FILEGROUP FG2
( NAME = N'VLTestDB2',
FILENAME =
N'c:\Apress\Recipes\VLTestDB2.ndf' ,
SIZE = 3048KB ,
FILEGROWTH = 1024KB ),
( NAME = N'VLTestDB3',
FILENAME =
N'c:\Apress\Recipes\VLTestDB3.ndf' ,
SIZE = 3048KB ,
FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'VLTestDB_log',
FILENAME =
N'c:\Apress\Recipes\VLTestDB_log.ldf' ,
SIZE = 1024KB ,
FILEGROWTH = 10%)
GO
```

805

第一个例子创建了一个文件组备份：

```
BACKUP DATABASE VLTestDB
FILEGROUP = 'FG2'
TO DISK = 'C:\Apress\Recipes\VLTestDB_FG2.bak'
```

它返回如下结果：

```
已为数据库'VLTestDB'，文件'VLTestDB2'(位于文件1上)处理了8页。
已为数据库'VLTestDB'，文件'VLTestDB3'(位于文件1上)处理了8页。
已为数据库'VLTestDB'，文件'VLTestDB_log'(位于文件1上)处理了3页。
BACKUP DATABASE...FILE=<name>成功处理了19页，花费0.082秒(1.756MB/秒)。
```

第二个例子演示了备份此数据库的两个指定文件的操作。为了先得到文件名称列表，执行sp_helpfile：

```
USE VLTestDB
GO
EXEC sp_helpfile
```

它返回如下部分结果：

```
name            fileid     filename
VLTestDB        1          c:\Apress\Recipes\VLTestDB.mdf
VLTestDB_log    2          c:\Apress\Recipes\VLTestDB_log.ldf
VLTestDB2       3          c:\Apress\Recipes\VLTestDB2.ndf
VLTestDB3       4          c:\Apress\Recipes\VLTestDB3.ndf
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使用sp_helpfile结果中的逻辑文件名，这个示例演示了备份TestDB数据库中TestDB3文件的操作：

```
BACKUP DATABASE VLTestDB
FILE = 'VLTestDB2',
FILE = 'VLTestDB3'
TO DISK = 'C:\apress\Recipes\VLTestDB_DB2_DB3.bak'
```

806

这个查询返回：

```
已为数据库'VLTestDB'，文件'VLTestDB2'(位于文件1上)处理了8页。
已为数据库'VLTestDB'，文件'VLTestDB3'(位于文件1上)处理了8页。
已为数据库'VLTestDB'，文件'VLTestDB_log'(位于文件1上)处理了2页。
BACKUP DATABASE...FILE=<name>成功处理了18页，花费0.278秒(0.499MB/秒)。
```

解析

此技巧从演示备份指定文件组的操作开始。这个语法除了需要指定FILEGROUP，几乎都和常规的完全数据库备份相同：

```
...
FILEGROUP = 'FG2'
...
```

第二个示例演示了使用FILE选项备份两个指定文件的操作，本例中备份了两个数据库文件：

```
...
FILE = 'VLTestDB2',
FILE = 'VLTestDB3'
...
```

从文件组或文件备份中还原的操作将在本章稍后进行演示。

29.2.12 执行部分备份

部分备份（partial backup）自动创建数据库中主文件组和所有激活读写功能的文件组的备份。这个选项对于那些存在只读文件组的超大型数据库是理想的，它不需要像那些可写的文件组备份得那么频繁。除了需要指定READ_WRITE_FILEGROUPS选项外，执行部分备份的语法几乎和完全备份的相同。如果想备份只读文件或文件组，也可以显式地指定它们。

为了准备进行这个示例，VLTestDB的FG2文件组将被设置为READONLY：

```
USE master
GO
ALTER DATABASE VLTestDB
MODIFY FILEGROUP FG2 READONLY
GO
```

返回的信息是：

807

```
文件组属性'READONLY'已设置。
```

现在，只读文件组被显式地包含在部分数据库备份中：

```
BACKUP DATABASE VLTestDB
READ_WRITE_FILEGROUPS
TO DISK = 'C:\Apress\Recipes\TestDB_Partial_include_FG3.bak'
```

返回的信息是：

```
已为数据库'VLTestDB'，文件'VLTestDB'(位于文件1上)处理了152页。
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
已为数据库'VLTestDB'，文件'VLTestDB_log'(位于文件1上)处理了1页。
BACKUP DATABASE...FILE=<name>成功处理了153页，花费0.120秒(9.960MB/秒)。
```

解析

只读文件组包含了不能写入的文件。因为只读数据不能改变，只需要周期性地对它进行备份（为了接受更新将它改为可读写的）。对于超大型数据库，对只读文件组做不必要的备份会耗尽时间和磁盘空间。使用新的部分数据库备份选项可以只对主文件组和所有可写文件组和文件进行备份操作，并不需要显式地列出所有的文件组。如果希望在备份中包含只读文件组，你仍然可以做到。

在这个技巧中，将FG2文件组设置为只读。然后使用READ_WRITE_FILEGROUPS选项备份数据库，这意味着FG2文件组中的文件不包含在备份中。

从部分备份进行数据库恢复的操作也假定你有对省略的文件进行的文件组或文件备份。本章稍后会演示从部分备份进行恢复操作。

29.2.13　查看备份元数据

一旦创建了备份，可以通过各种各样的RESTORE函数来查看媒体集的内容，包括RESTORE FILELISTONLY、RESTORE HEADERONLY、RESTORE VERIFYONLY和RESTORE LABELONLY。

- RESTORE LABLEONLY用来返回关于在指定备份设备上的备份媒体的信息。
- RESTORE HEADERONLY为所有创建在指定设备上的备份集返回一行。
- RESTORE FILELISTONLY更深层次地显示数据库文件名（逻辑的、物理的）以及其他备份数据库的信息。
- RESTORE VERIFYONLY预先验证备份设备以汇报RESTORE操作是否可以无错误地成功执行。

这4个命令的语法非常相似。在这个技巧中会演示这些命令的一般用法。

在大多数情况下，可以在编写实际的RESTORE DATABASE操作之前使用这些RESTORE命令去识别设备的内容。在此技巧的第一个示例中，从TestDB.bak设备中返回媒体集信息：

```
RESTORE LABELONLY
FROM DISK = 'C:\apress\Recipes\TestDB.bak'
```

它返回如下结果（部分）：

```
MediaName                   MediaDate
TestDB_October_2008_Week2   2008-10-14 16:23:30.000
```

808

在第二个查询中，检验了相同的设备，查看在它上面存在什么备份集：

```
RESTORE HEADERONLY
FROM DISK = 'C:\Apress\Recipes\TestDB.bak'
```

它返回如下结果（部分）：

```
Position    DatabaseName    DatabaseCreation            DateBackupTypeDescription
1           TestDB          2008-10-14 16:15:03.000     Database
```

在此技巧的第三个示例中，检验了在设备的备份集中的独立备份文件：

```
RESTORE FILELISTONLY
FROM DISK = 'C:\Apress\Recipes\TestDB.bak'
```

它返回如下结果（部分）：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
LogicalName     PhysicalName                    Type
TestDB          C:\Apress\Recipes\TestDB.mdf    D
TestDB_log      C:\Apress\Recipes\TestDB_log.LDF L
```

在此技巧中最后的示例中，验证了备份设备RESTORE的有效性：

```
RESTORE VERIFYONLY
FROM DISK = 'C:\Apress\Recipes\TestDB.bak'
WITH FILE = 1,
LOADHISTORY
```

这个查询返回：

```
文件1上的备份集有效。
```

解析

这4个命令RESTORE FILELISTONLY、RESTORE HEADERONLY、RESTORE VERIFYONLY和RESTORE LABELONLY对于在执行RESTORE操作之前收集所需要的信息时，每一个都是有用的。在此技巧的第一个示例中，使用RESTORE LABELONLY返回指定备份设备的媒体集信息。

第二个示例使用了RESTORE HEADERONLY查看在设备上实际存在什么备份集，因此当还原时可以指定要还原的备份集文件编号（也确保从正确的时期和备份类型中还原）。

此技巧第三个示例中使用了RESTORE FILELISTONLY返回备份在设备中的备份集中的实际的数据库文件。如果希望还原数据库到其他服务器，这是尤为有用的信息，因为新服务器和旧服务器对应的盘符和目录结构可能会不同。在本章稍后的技巧中，将学习如何在执行还原操作时移动数据库文件的位置。

最后的示例检验备份设备，确认它在RESTORE DATABASE操作中是否有效。通过使用FILE = 1指定了备份集。同时，通过使用LOADHISTORY选项将关于备份集的历史数据保存在msdb系统数据库中。 809

29.3 还原数据库

在本章的第一部分中集中介绍了如何备份数据库，包括如何执行完全备份、事务日志备份、差异备份以及文件和文件组备份。本章的第二部分将讲述如何从备份文件中还原数据库。还原操作将备份媒体集中的数据、日志、索引页复制到目标数据库。目标数据库可以是既有的数据库（它将会被覆盖）或新数据库（将创建基于备份的新文件）。在还原操作后，跟随了“重做”阶段，它前滚在数据库备份结束时已提交的事务。在它之后，“撤销”阶段回滚未提交的事务。

随后的一组技巧将演示数据库还原操作。

29.3.1 从完全备份还原数据库

在这个技巧中，将演示如何通过使用RESTORE命令从完全数据库备份中还原数据库。不像BACKUP操作那样，RESTORE不总是可以在线进行的操作——对于完全数据库还原来说，在还原数据库操作之前，用户连接必须要从数据库断开。其他还原类型（比如文件组、文件或页面）可以允许其他未受还原操作影响的地方的数据库中存在在线活动。例如，如果文件组FG2正在进行还原操作，FG3仍然可以在操作中访问。

注解 在线还原是SQL Server企业版的特性。

在常规的操作中，可能需要在用户错误引发的数据丢失、文件损坏、需要一份数据库的副本或移

29

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

动数据库到新的SQL Server实例中的情况下还原数据库。

随后是RESTORE命令的简要语法：

```
RESTORE DATABASE { database_name | @database_name_var }
[ FROM <backup_device> [ ,...n ] ]
[ WITH ] [Option Name ] [,...n]
```

RESTORE DATABASE命令还包括几个选项，其中大部分我会在本章中演示。

此技巧中第一个示例是一个从设备上最新的备份集中RESTORE的简单操作（在这个示例中，TestDB数据库在设备上存在两个备份集，并且你希望使用第二个备份集）。为了进行这个示范，要先在一个设备上创建两个完全备份：

```
BACKUP DATABASE TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Oct_15_2008.BAK'
GO

-- 过了一段时间，在这个设备上创建另一个备份

BACKUP DATABASE TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_Oct_15_2008.BAK'
GO
```

现在使用设备上的第二个备份还原数据库（注意，使用REPLACE参数告知SQL Server去覆盖既存的TestDB数据库）：

810

```
USE master
GO

RESTORE DATABASE TestDB
FROM DISK = 'C:\Apress\Recipes\TestDB_Oct_15_2008.bak'
WITH FILE = 2, REPLACE
```

它返回如下的输出：

```
已为数据库'TestDB'，文件'TestDB'(位于文件2上)处理了2456页。
已为数据库'TestDB'，文件'TestDB_log'(位于文件2上)处理了1页。
RESTORE DATABASE成功处理了2457页，花费5.578秒(3.607MB/秒)。
```

在第二个示例中，通过从TestDB备份中还原来创建新数据库，创建的新数据库叫做TrainingDB1。注意，使用MOVE参数去指定新数据库文件的位置：

```
USE master
GO

RESTORE DATABASE TrainingDB1
FROM  DISK = 'C:\Apress\Recipes\TestDB_Oct_15_2008.BAK'
WITH  FILE = 2,
MOVE 'TestDB' TO 'C:\Apress\Recipes\TrainingDB1.mdf',
MOVE 'TestDB_log' TO 'C:\Apress\Recipes\TrainingDB1_log.LDF'
```

这个查询返回：

```
已为数据库'TrainingDB1'，文件'TestDB'(位于文件2上)处理了2456页。
已为数据库'TrainingDB1'，文件'TestDB_log'(位于文件2上)处理了1页。
RESTORE DATABASE成功处理了2457页，花费4.799秒(4.193MB/秒)。
```

在此技巧最后的示例中，从条带化备份集中还原TestDB数据库（基于在本章前面创建的条带化备

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

份集）：

```
USE master
GO

RESTORE DATABASE TestDB
FROM DISK = 'C:\Apress\Recipes\TestDB_Stripe1.bak',
DISK = 'D:\Apress\Recipes\TestDB_Stripe2.bak',
DISK = 'E:\Apress\Recipes\TestDB_Stripe3.bak'
WITH FILE = 1, REPLACE
```

这个查询返回：

```
已为数据库'TestDB'，文件'TestDB'(位于文件1上)处理了152页。
已为数据库'TestDB'，文件'TestDB_log'(位于文件1上)处理了1页。
RESTORE DATABASE成功处理了153页，花费0.657秒(1.907MB/秒)。
```

811

解析

在第一个示例中，通过设置数据库到master数据库启动查询。这是因为完整RESTORE不是在线操作，而且为了运行还原操作，需要确认要还原的数据库上没有活动的连接。

RESTORE还原到TestDB数据库，并且将TestDB_Oct_15_2008.bak备份设备上的第二个备份集末尾的数据覆盖到当前的数据库。命令的第一行说明了要RESTORE的数据库：

```
RESTORE DATABASE TestDB
```

这个示例中的第二行指定了备份设备的位置：

```
FROM DISK = 'C:\Apress\Recipes\TestDB_Oct_15_2008.bak'
```

本示例中的最后一行指定RESTORE从哪个备份设备的备份集中进行（回顾本章前面讲到的，可以使用RESTORE HEADERONLY查看备份设备中有哪些备份集）：

```
WITH FILE = 2, REPLACE
```

在最后一次备份后更新的所有数据都将丢失，由此这个示例假定数据丢失是可接受的，并且恢复最后一次备份的数据。在第二个示例中，基于从另外的数据库的RESTORE操作创建了新数据库。这个示例与前面的查询相似，只是这次使用了MOVE命令去指定新数据库文件的位置（并且也使用了新的数据库名）：

```
MOVE 'TestDB' TO 'C:\apress\Recipes\TrainingDB1.mdf',
MOVE 'TestDB_log' TO 'C:\apress\Recipes\TrainingDB1_log.LDF'
```

可以使用RESTORE FILELISTONLY（前面演示过）返回备份数据库的逻辑名称和物理路径。

提示 RESTORE...MOVE命令经常被用来通过使用不同的盘符和目录，将数据库迁移到另外的SQL Server实例。

在此技巧中最后的示例中，从条带化备份集中还原TestDB。为集合中的每个磁盘设备重复运用了FROM DISK：

```
RESTORE DATABASE TestDB
FROM DISK = 'C:\apress\Recipes\TestDB_Stripe1.bak',
DISK = 'C:\apress\Recipes\TestDB_Stripe2.bak',
DISK = 'C:\apress\Recipes\TestDB_Stripe3.bak'
WITH FILE = 1, REPLACE
```

29

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在所有这些示例中，数据库被还原到了已恢复状态，意思是在重做阶段之后（并且在撤销阶段当中或之后），数据库对用户查询来说是在线的并且可用的。在随后的技巧中，将看到在差异或事务日志备份被还原之前，数据库经常不是已恢复状态。

29.3.2 从事务日志备份还原数据库

事务日志还原需要在完全数据库还原基础上进行，并且如果你应用了多个事务日志，必须按时间顺序应用它们（基于事务日志备份何时生成）。不按顺序应用事务日志或是在备份时有间隔都是不允许的。还原事务日志的语法用RESTORE LOG而不是RESTORE DATABASE，不过它们的语法和选项都是一样的。

812

为了建立这个示范，创建了一个名为TrainingDB的新数据库：

```
IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'TrainingDB')
BEGIN
    CREATE DATABASE TrainingDB
END
GO

-- 新建一张表并且添加一些数据
USE TrainingDB
GO

SELECT *
INTO dbo.SalesOrderDetail
FROM AdventureWorks.Sales.SalesOrderDetail
GO
```

数据库将执行一个完全备份和两个连续的事务日志备份：

```
BACKUP DATABASE TrainingDB
TO DISK = 'C:\Apress\Recipes\TrainingDB.bak'
GO

BACKUP LOG TrainingDB
TO DISK = 'C:\Apress\Recipes\TrainingDB_Oct_14_2008_8AM.trn'
GO

-- 经过两个小时后，运行了另外一个事务日志备份

BACKUP LOG TrainingDB
TO DISK = 'C:\Apress\Recipes\TrainingDB_Oct_14_2008_10AM.trn'
GO
```

第一个RESTORE示例假定要还原到的数据库没有存在连接。但是，在示例中我还是演示了在执行RESTORE之前如何踢掉所有在数据库上的连接：

```
USE master
GO

-- 踢掉所有其他连接
ALTER DATABASE TrainingDB
SET SINGLE_USER
WITH ROLLBACK IMMEDIATE
```

下一步，从备份中还原一个数据库备份和两个事务日志备份：

```
RESTORE DATABASE TrainingDB
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FROM DISK = 'C:\Apress\Recipes\TrainingDB.bak'
WITH NORECOVERY, REPLACE

RESTORE LOG TrainingDB
FROM DISK = 'C:\Apress\Recipes\TrainingDB_Oct_14_2008_8AM.trn'
WITH NORECOVERY, REPLACE

RESTORE LOG TrainingDB
FROM DISK = 'C:\Apress\Recipes\TrainingDB_Oct_14_2008_10AM.trn'
WITH RECOVERY, REPLACE
```

这个查询返回：

```
已为数据库'TrainingDB'，文件'TrainingDB'(位于文件1上)处理了1656页。
已为数据库'TrainingDB'，文件'TrainingDB_log'(位于文件1上)处理了2页。
RESTORE DATABASE成功处理了1658页，花费4.164秒(3.260MB/秒)。
已为数据库'TrainingDB'，文件'TrainingDB'(位于文件1上)处理了0页。
已为数据库'TrainingDB'，文件'TrainingDB_log'(位于文件1上)处理了2页。
RESTORE LOG成功处理了2页，花费0.066秒(0.186MB/秒)。
RESTORE LOG成功处理了0页，花费0.072秒(0.000MB/秒)。
```

在第二个示例中，使用STOPAT去还原数据库和事务日志到指定的时间点。为了演示它，首先要做一个TrainingDB数据库的完全备份：

```
BACKUP DATABASE TrainingDB
TO DISK = 'C:\Apress\Recipes\TrainingDB_Oct_14_2008.bak'
```

下一步，从表中删除一些数据，然后查询在删除操作后的当前时间：

```
USE TrainingDB
GO

DELETE dbo.SalesOrderDetail
WHERE ProductID = 776
GO

SELECT GETDATE()
GO
```

这个查询返回：

```
2008-10-14 20:20:56.583
```

下一步，执行事务日志备份：

```
BACKUP LOG TrainingDB
TO DISK = 'C:\Apress\Recipes\TrainingDB_Oct_14_2008_2022.trn'
```

这个查询返回：

```
已为数据库'TrainingDB'，文件'TrainingDB_log'(位于文件1上)处理了18页。
BACKUP LOG成功处理了18页，花费0.163秒(0.876MB/秒)。
```

下一步，从备份中还原数据库，为了也可以还原事务日志备份，我们使用了NORECOVERY选项：

```
USE master
GO

RESTORE DATABASE TrainingDB
FROM DISK = 'C:\Apress\Recipes\TrainingDB_Oct_14_2008.bak'
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
WITH FILE = 1, NORECOVERY,
814 STOPAT = '2008-10-14 20:18:56.583'
GO
```

下一步，还原事务日志，还指定了在数据删除之前的时间：

```
RESTORE LOG TrainingDB
FROM DISK = 'C:\Apress\Recipes\TrainingDB_Oct_14_2008_2022.trn'
WITH RECOVERY,
STOPAT = '2008-10-14 20:18:56.583'
GO
```

下一步，随后的查询确认你还原的数据库恰好发生在数据删除操作之前：

```
USE TrainingDB
GO

SELECT COUNT(*)
FROM dbo.SalesOrderDetail
WHERE ProductID = 776
GO
```

这个查询返回：

```
228
```

解析

在此技巧的第一个示例中，从完全数据库备份中还原TrainingDB数据库，并将其置为NORECOVERY模式。置为NORECOVERY模式允许附加其他事务日志或差异备份。在这个示例中，按照时间顺序附加了两个事务日志备份，在使用RECOVERY选项的那一瞬间，数据库置为联机状态。

此技巧的第二个示例演示了将数据库还原到一个指定时间点的操作。时点恢复在将数据库还原到数据库发生修改或是故障之前的时候是有用的。语法与前一个示例中的类似，只是在RESTORE DATABASE和RESTORE LOG中使用了STOPAT。每个RESTORE语句包含STOPAT确保了还原操作不会越过指定的时间恢复。

29.3.3 从差异备份还原数据库

差异数据库还原的语法与完全数据库还原的相同，只是完全数据库还原必须要执行在附加差异备份之前。当还原完全数据库备份时，数据库必须置为NORECOVERY模式。而且任何希望还原的事务日志必须在附加了差异备份之后完成，正如示例所演示的。

首先，这个示例先对TrainingDB数据库执行完全备份、差异备份、事备日志备份：

```
USE master
GO

BACKUP DATABASE TrainingDB
TO DISK = 'C:\Apress\Recipes\TrainingDB_DiffExample.bak'

-- 过了一段时间

BACKUP DATABASE TrainingDB
TO DISK = 'C:\Apress\Recipes\TrainingDB_DiffExample.diff'
815 WITH DIFFERENTIAL

-- 又过了一段时间
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
BACKUP LOG TrainingDB
TO DISK = 'C:\Apress\Recipes\TrainingDB_DiffExample_tlog.trn'
```

现在，将演示执行RESTORE，将数据库还原到最后事务日志备份完成时的状态：

```
USE master
GO

-- 完全数据库备份
RESTORE DATABASE TrainingDB
FROM DISK = 'C:\Apress\Recipes\TrainingDB_DiffExample.bak'
WITH NORECOVERY, REPLACE

-- 差异备份
RESTORE DATABASE TrainingDB
FROM DISK = 'C:\Apress\Recipes\TrainingDB_DiffExample.diff'
WITH NORECOVERY

-- 事务日志备份
RESTORE LOG TrainingDB
FROM DISK = 'C:\Apress\Recipes\TrainingDB_DiffExample_tlog.trn'
WITH RECOVERY
```

这个查询返回：

```
已为数据库'TrainingDB'，文件'TrainingDB'(位于文件1上)处理了152页。
已为数据库'TrainingDB'，文件'TrainingDB_log'(位于文件1上)处理了2页。
RESTORE DATABASE成功处理了154页，花费0.443秒(2.831MB/秒)。
已为数据库'TrainingDB'，文件'TrainingDB'(位于文件1上)处理了40页。
已为数据库'TrainingDB'，文件'TrainingDB_log'(位于文件1上)处理了1页。
RESTORE DATABASE成功处理了41页，花费0.069秒(4.860MB/秒)。
RESTORE LOG成功处理了0页，花费0.070秒(0.000MB/秒)。
```

解析

差异备份捕捉数据库在最后一次完全数据库备份之后发生的变化。差异还原使用了与完全数据库还原相同的语法，只是它们必须总是在完全数据库还原（设置NORECOVERY）之后。在此技巧中，数据库最初从完全数据库还原，然后是差异备份的还原，最后是事务日志备份的还原。差异RESTORE命令的组成与前面RESTORE示例中的相似，只不过它是基于差异备份文件的。在最后一次还原操作时，指定RECOVERY选项使数据库处于可用状态。

29.3.4 还原文件或文件组

还原文件或文件组几乎使用了和完全数据库还原相同的语法，只是还使用了FILEGROUP或FILE关键字。为了执行一个指定可读写文件或文件组的还原操作，数据库必须设置为FULL或BULK恢复模式。这是因为附加事务日志备份的操作必须在还原文件或文件组备份之后运行。在SQL Server中，如果数据库设置为简单恢复模式，那么只有只读文件或只读文件组才可进行文件 / 文件组备份和还原。

此技巧的示例先对VLTestDB数据库进行文件组备份：

816

```
USE master
GO

BACKUP DATABASE VLTestDB
FILEGROUP = 'FG2'
TO DISK = 'C:\Apress\Recipes\VLTestDB_FG2.bak'
WITH NAME = N'VLTestDB-Full Filegroup Backup',
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
SKIP, STATS = 20
GO
```

过一段时间后，对数据库进行事务日志备份：

```
BACKUP LOG VLTestDB
TO DISK = 'C:\Apress\Recipes\VLTestDB_FG_Example.trn'
```

下一步，从备份中还原数据库的FG2文件组，然后还原事务日志备份：

```
USE master
GO

RESTORE DATABASE VLTestDB
FILEGROUP = 'FG2'
FROM DISK = 'C:\Apress\Recipes\VLTestDB_FG2.bak'
WITH FILE = 1, NORECOVERY, REPLACE

RESTORE LOG VLTestDB
FROM DISK = 'C:\Apress\Recipes\VLTestDB_FG_Example.trn'
WITH FILE = 1, RECOVERY
```

这个查询返回：

```
已为数据库'VLTestDB'，文件'VLTestDB2'(位于文件1上)处理了8页。
已为数据库'VLTestDB'，文件'VLTestDB3'(位于文件1上)处理了8页。
RESTORE DATABASE ... FILE=<name>成功处理了16页，花费0.119秒(1.101MB/秒)。
已为数据库'VLTestDB'，文件'VLTestDB2'(位于文件1上)处理了0页。
已为数据库'VLTestDB'，文件'VLTestDB3'(位于文件1上)处理了0页。
RESTORE LOG成功处理了0页，花费0.062秒(0.000MB/秒)。
```

解析

文件组或文件备份经常在非常大的数据库中使用，对它进行完全数据库备份可能会花费非常长的时间。使用文件组或文件备份会增大管理的复杂度，因为你必须在事故发生后使用多个备份集去进行恢复（比如说每个文件组一个备份集）。

在这个技巧中，从备份设备中还原名为FG2的VLTestDB数据库文件组，并且为了可以附加事务日志还原操作设为NORECOVERY模式。为了将文件组恢复到联机状态，在事务日志还原操作使用了RECOVERY关键字。在SQL Server企业版中，不同于主文件组的文件组可以在还原操作中被置为离线状态，保持其他活动文件组为可用状态（这被称为ONLINE还原）。

817

29.3.5 执行部分（PARTIAL）还原

在段落还原方式中，可以在RESTORE DATABASE命令中使用PARTIAL命令去还原辅助文件组。这个RESTORE的变种将主文件组置为联机状态，然后在有需要的情况下还原其他文件组。如果使用置为完整或大容量日志恢复模式的数据库，则可以在可读写文件组上使用这个命令。如果数据库置为简单恢复模式，则只能在只读辅助文件组上使用PARTIAL。

在这个示例中，通过使用PARTIAL关键字从完全数据库备份中还原VLTestDB，并且指定只有PRIMARY文件组置为在线状态（同时FG2和FG3文件组保持离线和未还原状态）。

首先，为了启动这个示例，备份VLTestDB中的主文件组和FG2文件组：

```
USE master
GO

BACKUP DATABASE VLTestDB
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
FILEGROUP = 'PRIMARY'
TO DISK = 'C:\Apress\Recipes\VLTestDB_Primary_PieceExmp.bak'
GO

BACKUP DATABASE VLTestDB
FILEGROUP = 'FG2'
TO DISK = 'C:\Apress\Recipes\VLTestDB_FG2_PieceExmp.bak'
GO
```

在此之后，执行事务日志备份：

```
BACKUP LOG VLTestDB
TO DISK = 'C:\Apress\Recipes\VLTestDB_PieceExmp.trn'
GO
```

下一步，执行部分RESTORE，只恢复PRIMARY文件组：

```
RESTORE DATABASE VLTestDB
FILEGROUP = 'PRIMARY'
FROM DISK = 'C:\Apress\Recipes\VLTestDB_Primary_PieceExmp.bak'
WITH PARTIAL, NORECOVERY, REPLACE

RESTORE LOG VLTestDB
FROM DISK = 'C:\Apress\Recipes\VLTestDB_PieceExmp.trn'
WITH RECOVERY
```

另一个文件组FG2现在包含不可用的文件。可以通过从VLTestDB数据库查询sys.database_files查看文件的状态：

```
USE VLTestDB
GO

SELECT name,
   state_desc
FROM sys.database_files
```

这个查询返回：

818

```
Name                state_desc
VLTestDB            ONLINE
VLTestDB_log        ONLINE
VLTestDB2           RECOVERY_PENDING
VLTestDB3           RECOVERY_PENDING
```

解析

在这个技巧中，从完全备份中还原VLTestDB，只还原了PRIMARY文件组。WITH子句包含了PARTIAL关键字和NORECOVERY，由此可以还原事务日志备份。在事务日志还原之后，所有在PRIMARY文件组中的对象都将可用，并且在还原辅助文件组之前，在辅助文件组中的对象都是不可用的。

对于非常大的数据库，在RESTORE操作中使用PARTIAL关键字允许以优先顺序排列文件组并载入那些高优先级的文件组，使它们更快地变为可用。

29.3.6 还原页面

SQL Server提供了对置为FULL或BULK_LOGGED恢复模式的数据库还原指定数据页的能力。在很少发生的、数据库中少许数据页被破坏的情况下，还原单个数据页可能比还原整个文件、文件组或整个数据库更为有效。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

还原指定页面的语法与还原文件组或还原数据库相似，只是需要使用PAGE关键字和页ID。在msdb.dbo.suspect_pages系统表、SQL错误日志或DBCC命令输出的返回中会标识出那些坏的页面。

这个示例先创建了TestDB数据库的完全数据库备份：

```
BACKUP DATABASE TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_PageExample.bak'
GO
```

下一步，使用PAGE参数执行还原操作：

```
RESTORE DATABASE TestDB
PAGE='1:8'
FROM DISK = 'C:\Apress\Recipes\TestDB_PageExample.bak'
WITH NORECOVERY, REPLACE
GO
```

这个查询返回：

```
已为数据库'TestDB'，文件'TestDB'(位于文件1上)处理了1页。
RESTORE DATABASE ... FILE=<name>成功处理了1页，花费1.107秒(0.007MB/秒)。
```

在这一点上，可以对在最后完全备份之后产生的任何差异备份或事务日志备份进行还原操作。在这个示例中没有进行这些操作，没有继续还原其他备份。

下一步，进行和前一示例相关的操作，必须创建新的事务日志备份来捕捉还原的页面：

```
BACKUP LOG TestDB
TO DISK = 'C:\Apress\Recipes\TestDB_PageExample_tlog.trn'
GO
```

819

这个查询返回：

```
已为数据库'TestDB'，文件'TestDB_log'(位于文件1上)处理了2页。
BACKUP LOG成功处理了2页，花费0.840秒(0.014MB/秒)。
```

为了完成页面还原操作，在RESTORE...PAGE之后必须执行带RECOVERY的事务日志还原操作：

```
RESTORE LOG TestDB
FROM DISK = 'C:\Apress\Recipes\TestDB_PageExample_tlog.trn'
WITH RECOVERY
```

解析

在这个技巧中，通过在RESTORE DATABASE命令中使用PAGE选项从完全数据库备份中还原单个数据页。正如从FILE或FILEGROUP中还原，第一个RESTORE将数据库置为NORECOVERY状态，这将允许在完成还原之前附加额外的事务日志备份。

29.3.7　用多种恢复路径识别数据库

当使用时点恢复从备份中恢复数据库，或不恢复最新的差异备份或事务备份链来恢复数据库时，可以使用多种恢复路径。如果存在不用在RESTORE处理中的备份，就在恢复路径中创建了分叉。

这个技巧定示如何使用sys.database_recovery_status目录视图来获取关于存在一个以下恢复路径数据库的信息。第一步，将创建新的数据库并对它执行完全数据库备份，创建表并插入一些行，然后以事务日志备份结束：

```
USE master
GO
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
IF NOT EXISTS (SELECT name
               FROM sys.databases
               WHERE name = 'RomanHistory')
BEGIN
    CREATE DATABASE RomanHistory
END
GO

BACKUP DATABASE RomanHistory
TO DISK = 'C:\Apress\RomanHistory_A.bak'
GO

USE RomanHistory
GO

CREATE TABLE EmperorTitle
   (EmperorTitleID int NOT NULL PRIMARY KEY IDENTITY(1,1),
    TitleNM varchar(255))
GO

INSERT EmperorTitle (TitleNM)
VALUES ('Aulus'), ('Imperator'), ('Pius Felix'), ('Quintus')

BACKUP LOG RomanHistory
TO DISK = 'C:\Apress\RomanHistory_A.trn'
GO
```

820

接着，查询目录视图来获取在这个时间点数据库的信息（在本书中为了适合阅读，列别名进行了缩减）：

```
SELECT last_log_backup_lsn LastLSN, recovery_fork_guid Rec_Fork,
first_recovery_fork_guid Frst_Fork, fork_point_lsn Fork_LSN
FROM sys.database_recovery_status
WHERE database_id = DB_ID('RomanHistory')
```

这个查询返回如下内容（你的结果将会不同）：

LastLSN	Rec_Fork	Frst_Fork	Fork_LSN
18000000010900001	0020752F-1085-49F6-A848-21C9EDBFF290	NULL	NULL

注意first_recovery_fork_guid和fork_point_lsn为NULL。这是因为在这个恢复路径中还没有创建分叉。last_log_backup_lsn字段表示最新的日志更新的LSN，recovery_fork_guid显示了数据库中处于活动状态的当前恢复路径。

提示 日志序列号（LSN）唯一标识数据库事务日志的每个记录。

下一步，执行几个数据修改和另一个事务日志备份：

```
INSERT EmperorTitle (TitleNM)
VALUES ('Germanicus'), ('Lucius'), ('Maximus'), ('Titus')

BACKUP LOG RomanHistory
TO DISK = 'C:\Apress\RomanHistory_B.trn'
GO
```

现在将继续将数据库RESTORE到之前的状态（但不是最新的状态）：

```
USE master
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
GO

RESTORE DATABASE RomanHistory
FROM DISK = 'C:\Apress\RomanHistory_A.bak'
WITH NORECOVERY

RESTORE DATABASE RomanHistory
FROM DISK = 'C:\Apress\RomanHistory_A.trn'
WITH RECOVERY
```

现在如果对前面的查询查看sys.database_recovery_status，将会看到fork_point_lsn和first_recovery_fork_guid列都不再为NULL：

```
SELECT last_log_backup_lsn LastLSN, recovery_fork_guid Rec_Fork,
first_recovery_fork_guid Frst_Fork, fork_point_lsn Fork_LSN
FROM sys.database_recovery_status
WHERE database_id = DB_ID('RomanHistory')
```

821

这个查询返回：

LastLSN	Rec_Fork	Frst_Fork	Fork_LSN
18000000010900001	F18522D8-6FDB-40BE-AB99-047DE4280F40	D020752F-1085-49F6-A848-21C9EDBFF290	18000000010900001

解析

使用sys.database_recovery_status目录视图可以查看数据库是否存在多个恢复路径。

在这个技巧中，我做了完全数据库备份以及两个事务日志备份。如果使用所有3个备份来还原数据库，我将保持在同一个恢复路径中。但是，我只还原了第一个完全备份以及第一个事务日志备份，将数据库置于还原第二个事务日志之前的恢复状态。通过提前恢复，我将数据库以第二个恢复路径置为联机状态。

822

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

索　引

索引中页码为英文原书页码，与书中边栏页码一致。

Symbols

@ prefix（@前缀），13，328
@@ prefix（@@前缀），293
%= modulo, assign operator（取模等于，赋值运算符），84
&= bitwise &, assign operator（位与等于，赋值运算符），84
^= bitwise exclusive OR, assign operator（位异或等于，赋值运算符），84
|= bitwise |, assign operator（位或等于，赋值运算符），84
.bak files（.bak文件），794
*= multiply, assign operator（乘等于，赋值运算符），84
+= add, assign operator（加等于，赋值运算符），84
-= subtract, assign operator（减等于，赋值运算符），84
/= divide, assign operator（除等于，赋值运算符），84
= equality operator（等于运算符），84
; semicolon（分号），59
[] wildcard（[]通配符），11
[^] wildcard（[^]通配符），11
!< operator（!<运算符），8
!> operator（!>运算符），8
!= operator（!=运算符），8
sign，prefixing local temporary tables（#符号，局部临时表前缀），176
sign，prefixing global temporary tables（##符号，全局临时表前缀），176
% wildcard（%通配符），11
< operator（<运算符），8
<> operator（<>运算符），8
<= operator（<=运算符），8
> operator（>运算符），8
+ operator（+运算符），24
= operator（=运算符），8
_ wildcard（_通配符），11

A

ABS function（ABS函数），261
Accent sensitivity（区分重音），218
ACID test (Atomicity, Consistency, Isolation (or Independence), and Durability)（ACID测试（原子性、一致性、隔离性（独立性）以及持久性）），115
ACOS function（ACOS函数），261
ad hoc queries，executing via OPENROWSET command（即席查询，通过OPENROWSET命令执行），733
add, assign (+=) operator（加等于，赋值运算符），84
ADD FILE command（ADD FILE命令），654
ADD FILEGROUP command（ADD FILEGROUP命令），660
ADD LOG FILE command（ADD LOG FILE命令），655
AdventureWorks sample database, downloading,（AdventureWorks示例数据库，下载），1
AFTER DML triggers（AFTER DML触发器），375
AFTER triggers（AFTER触发器），374
aggregate functions（聚合函数），22，257–261
aggregated performance statistics, viewing（聚合的性能统计信息，查看），750
alias data types（别名数据类型）. 参见 user-defined types
aliases，732
 column （列），15，22
 table（表），29
ALL operator（ALL运算符），8
allocated unit lock resource（分配单元锁资源），124
ALLOW_SNAPSHOT_ISOLATION database option（ALLOW_SNAPSHOT_ISOLATION数据库选项），647，648
ALTER APPLICATION ROLE command（ALTER APPLICATION ROLE命令），498
ALTER ASSEMBLY command（ALTER ASSEMBLY命令），417
ALTER ASYMMETRIC KEY command（ALTER ASYMMETRIC KEY命令），557
ALTER BROKER PRIORITY command（ALTER BROKER PRIORITY命令），597
ALTER CERTIFICATE command（ALTER CERTIFICATE命令），570
ALTER COLUMN command（ALTER COLUMN命令），149
ALTER DATABASE AUDIT SPECIFICATION command（ALTER DATABASE AUDIT SPECIFICATION命令），543
ALTER DATABASE command（ALTER DATABASE命令），213, 393, 397, 581,629, 631
 ANSI SQL options and（ANSI SQL选项），637
 changing recovery mode and（修改恢复模式），649
 cursor options and（游标选项），642
 database compatibility level and（数据库兼容级别），623
 database mirroring and（数据库镜像），700，707
 database state and（数据库状态），652
 external access and（外部访问），640

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

files/filegroups and（文件 / 文件组），654–663
operating modes，configuring via（操作模式，配置），711
page verification and（页验证），651
parameterization and（参数化），644
read-consistency options and（读一致性选项），648
shrinking databases/database files and（收缩数据库 / 数据库文件），667
Transparent Data Encryption and（透明数据加密），575
ALTER DATABASE...SET PARTNER FAILOVER command（ALTER DATABASE...SET PARTNER FAILOVER 命令），712
ALTER DATABASE...SET PARTNER FORCE_SERVICE_ALLOW_DATA_LOSS command（ALTER DATABASE...SET PARTNER FORCE_SERVICE_ALLOW_DATA_LOSS命令），712
ALTER DATABASE...SET PARTNER OFF command（ALTER DATABASE...SET PARTNER OFF命令），714
ALTER DATABASE...SET PARTNER SUSPEND command（ALTER DATABASE...SET PARTNER SUSPEND命令），713
ALTER DATABASE...SET PARTNER TIMEOUT command（ALTER DATABASE...SET PARTNER TIMEOUT命令），714
ALTER FULLTEXT CATALOG command（ALTER FULLTEXT CATALOG命令），221
ALTER FULLTEXT INDEX command（ALTER FULLTEXT INDEX命令），222
ALTER FULLTEXT STOPLIST command（ALTER FULLTEXT STOPLIST命令），227, 229
ALTER FUNCTION command（ALTER FUNCTION命令），298
ALTER INDEX command（ALTER INDEX命令），200, 205, 215, 429
ALTER INDEX REBUILD command（ALTER INDEX REBUILD命令），682
ALTER INDEX REORGANIZE command（ALTER INDEX REORGANIZE命令），682, 685
ALTER LOGIN command（ALTER LOGIN命令），483, 612
ALTER MASTER KEY command（ALTER MASTER KEY命令），552, 554
ALTER OWNERSHIP command（ALTER OWNERSHIP命令），527
ALTER PARTITION FUNCTION command（ALTER PARTITION FUNCTION命令），186
ALTER PARTITION SCHEME command（ALTER PARTITION SCHEME命令），186
ALTER PROCEDURE command（ALTER PROCEDURE命令），332
ALTER QUEUE command（ALTER QUEUE命令），598
ALTER RESOURCE GOVERNOR command（ALTER RESOURCE GOVERNOR命令），786
ALTER RESOURCE POOL command（ALTER RESOURCE POOL命令），784
ALTER ROLE command（ALTER ROLE命令），495
ALTER SCHEMA command（ALTER SCHEMA命令），517, 689
ALTER SCHEMA TRANSFER command（ALTER SCHEMA TRANSFER命令），689
ALTER SERVER AUDIT command（ALTER SERVER AUDIT命令），539, 542, 544
ALTER SERVER AUDIT SPECIFICATION command（ALTER SERVER AUDIT SPECIFICATION命令），543
ALTER SYMMETRIC KEY command（ALTER SYMMETRIC KEY命令），562
ALTER TABLE command（ALTER TABLE命令），147–154,175
Change Tracking and（更改跟踪），109
constraints and（约束）159, 170, 172, 175
foreign keys and（外键），161
lock escalation and（锁扩散），126
moving partitions between tables（在表之间移动分区），189
rebuilding heaps and（重建堆），686
table column collation and（表列排序规则），156
ALTER TRACE permission（ALTER TRACE权限），505
ALTER TRIGGER command（ALTER TRIGGER命令），393
ALTER USER WITH LOGIN command（ALTER USER WITH LOGIN命令），492
ALTER USER command（ALTER USER命令），490
ALTER USER WITH LOGIN command（ALTER USER WITH LOGIN命令），492
ALTER VIEW command（ALTER VIEW命令），245, 247
ALTER WORKLOAD GROUP command（ALTER WORKLOAD GROUP命令），785
ALTER XML SCHEMA COLLECTION command（ALTER XML SCHEMA COLLECTION命令），422
anchor members（锚成员），61
ANSI SQL options（ANSI SQL选项），636–638
ANY operator（ANY运算符），8
application lock resource（应用程序锁资源），124
application roles（应用程序角色），488, 497–499
APPLY clause（APPLY子句），35–38
APP_NAME function（APP_NAME函数），301
artificial keys（人工键），165–168
ASCII files，importing（ASCII文件，导入），735–738
ASCII function（ASCII函数），264
ASIN function（ASIN函数），261
assemblies（程序集），402
administering（管理），417
compiling into DLL files（编译为DLL文件），408
loading into SQL Server（加载到SQL Server），409
security and（安全性），406
when to use/not to use（当使用或不使用时），402
assignment operators（赋值运算符），84–86
asymmetric key encryption（非对称密钥加密），555–560
asymmetric keys（非对称密钥），547
asynchronous processing（异步处理），579
ATAN function（ATAN函数），261
ATN2 function（ATN2函数），261
atomicity（原子性），115
Atomicity，Consistency，Isolation (or Independence)，and Durability (ACID test)（原子性、一致性、隔离性（独立性）以及持久性（ACID测试）），117
audit action groups（审核活动组），536

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

audit events（审核事件），536
auditing SQL instance- /database-scoped activity（审核SQL实例范围/数据库范围的活动），501, 529–545
authentication，mixed（混和身份验证），476, 480
authorization（授权），480
AUTO option（AUTO选项），127
autocommit transactions（自动提交事务），116
automatic database options（自动数据库选项），638
automatic page recovery（自动页恢复），697
AUTO_CREATE_STATISTICS database option（AUTO_CREATE_STATISTICS数据库选项），740,760
AUTO_UPDATE_STATISTICS database option（AUTO_UPDATE_STATISTICS数据库选项），760
AVG aggregate function（AVG聚合函数），257,258

B

B-tree structures（B树结构），198
BACKUP CERTIFICATE command（BACKUP CERTIFICATE命令），569
BACKUP command（BACKUP命令），793, 797, 799
BACKUP DATABASE command（BACKUP DATABASE命令），793, 805
backup device definitions（备份设备定义），799
BACKUP LOG command（BACKUP LOG命令），803
BACKUP MASTER KEY command（BACKUP MASTER KEY命令），553
backup sets（备份集），796
 retention options for（保持期选项），797
 striping（条带化），799
backups（备份），789–809
 compression for（压缩），794
 differential（差异备份），792, 805–809
 full（完整备份），791, 793–804
 naming/describing（命名/描述），796
 partial（部分），807
 plan for（计划），789–791
 sequence of（序列），792, 804
 striping（条带化），799
 of transaction logs（事务日志），791, 803
BEGIN CATCH command（BEGIN CATCH命令），470
BEGIN DIALOG CONVERSATION command（BEGIN DIALOG CONVERSATION命令），589
BEGIN DISTRIBUTED TRANSACTION command（BEGIN DISTRIBUTED TRANSACTION命令），116
BEGIN...END statement（BEGIN…END语句），312
BEGIN TRANSACTION command（BEGIN TRANSACTION命令），116
BEGIN TRY command（BEGIN TRY命令），469
best practices（最佳实践）
 for assemblies, grouping related functionalities within（将相关功能集合到程序集），406
 for BEGIN...END statement（BEGIN...END语句），312
 for Change Tracking（更改跟踪），108
 for control-of-flow（控制流），318
 for documenting stored procedures（为存储过程写文档），335
 for data type conversions（数据类型转换），28
 for MAXRECURSION hint（MAXRECURSION提示），62
 for views（视图），240, 246
 for Snapshot Isolation（快照隔离），113
BETWEEN operator（BETWEEN运算符），8, 9
binary files，importing（二进制文件，导入），735–738
bitwise &, assign (&=) operator（位与等于，赋值运算符），84
bitwise exclusive OR, assign (^=) operator（位异或等于，赋值运算符），84
bitwise |, assign (|=) operator（位或等于，赋值运算符），84
blocking（阻塞），134–137
bottlenecks，（瓶颈）752
built-in error messages（内建错误消息），459–462
built-in functions（内建函数）. 参见 functions
BULK option（BULK选项），79, 723, 735–738
bulk update locks（大容量更新锁），123
BULK_LOGGED recovery model（BULK_LOGGED恢复模式），649, 791

C

cached query plans（缓存的查询计划），327, 340, 748
caller permissions（调用者权限），337
Cartesian products（笛卡尔积），29, 32
cascading changes feature（级联修改特性），163
case, uppercase/lowercase conversions and（大小写转换），346–349
case sensitivity/insensitivity，collations and（区分/不区分大小写，排序规则），641
CASE statement（CASE语句），307–310
CAST function（CAST函数），284–288
catalogs，full-text（目录，全文），217–230
CDC (Change Data Capture)（变更数据捕获），93, 96–107
 disabling from tables/database（在表和数据库中禁用），107
 enabling/configuring（启用/配置），96
CDC system tables（CDC系统表），99–104
CDC update mask（CDC更新掩码），104
CEILING function（CEILING函数），261
certificate encryption（证书加密），567–573
Change Data Capture. *See* CDC（变更数据捕获，参见CDC）
Change Tracking（更改跟踪），93, 107–113
 enabling（启用），108
 disabling for full-text indexes（在全文索引中禁用），224
CHAR function（CHAR函数），264
character data types（字符数据类型），146
character values，converting to/from ASCII（字符值，从ASCII转化，或转化为ASCII），264
characters，string functions for（字符，字符串函数），263–274
CHARINDEX function（CHARINDEX函数），266, 269
CHECK constraint（CHECK约束），168, 171–174, 251, 255
CHECKSUM option（CHECKSUM选项），650
CHECKSUM_AGG aggregate function（CHECKSUM_AGG

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

聚合函数），257
CHECK_POLICY option（CHECK_POLICY选项），482
chunking data（数据分块），89
CLOSE MASTER KEY command（CLOSE MASTER KEY命令），554
CLOSE SYMMETRIC KEY command（CLOSE SYMMETRIC KEY命令），562
CLR (Common Language Runtime)（CLR公共语言运行时），325, 401
CLR data type to SQL Server data type translations（CLR数据类型到SQL Server数据类型的转化），407
CLR database objects，creating（CLR数据库对象，创建），404–416
CLR integration（CLR集成），401–418
CLR objects（CLR对象），404
CLR scalar user-defined functions，creating（CLR标量用户定义函数，创建），412–414
CLR stored procedures creating（创建CLR存储过程），410
 writing assemblies for（写程序集），405
CLR triggers，creating（CLR触发器，创建），415–416
clustered indexes（聚集索引），198, 755
COALESCE function（COALESCE函数），274, 276
code pages，collations for（代码页，排序规则），155
COLLATE command（COLLATE命令），155, 641
collations（排序规则），155, 641
column aliases（列别名），15, 22
column constraints（列约束），157
column sets，（列集）153
columns（列），147–154
 computed（计算列），149
 constraints and（约束），168–176
 data types and（数据类型），144
 dropping（删除），153
 grouping（分组），14–17
 identity of，checking（标识，检查），166
 mapping to partitions（映射到分区），180
 renaming（重命名），22, 687–689
 rows，converting to via UNPIVOT（行，通过UNPIVOT转化），42
 SELECT statement and（SELECT语句），21–27
 selecting for rows（选择行），2
 sparse（稀疏列），150–153
 uniqueidentifier（唯一标识符），67
 user-defined types and，identifying（用户定义类型，标识出），367
 values，converting into（值，转化为），39
 xml data types for（xml数据类型），419
 XML data，inserting into（XML数据，插入到），421
column sets（列集），151
comma delimited lists，creating via SELECT statement（逗号分隔列表，通过SELECT语句创建），25
COMMIT TRANSACTION command（COMMIT TRANSACTION命令），116
Common Language Runtime (CLR)（公共语言运行时（CLR），参见CLR下的条目）
comparisons（比较），9
compatibility levels，for databases（兼容级别，数据库），623
composite indexes（组合索引），199, 202
composite primary keys（组合主键），157
compound assignment operators（组合赋值运算符），84–86
compression（压缩）
 for backups（用于备份），794
 for indexes/partitions（用于索引和分区），212
COMPUTE BY clause (deprecated with SQL Server 2005)（COMPUTE BY子句（SQL Server 2005不建议使用）），48
computed columns（计算列），149
{CONCAT | HASH | MERGE} UNION hint（{CONCAT | HASH | MERGE} UNION提示），451
concurrency（并发性），123, 128, 129
 improving for indexes（改进索引），209
 indexing locking and（索引锁定），211
 lock activity and（锁定活动），126
conditional processing（条件处理），307–312
configurations, query performance and（配置，查询性能），739
CONNECT permissions（CONNECT权限），605
connection timeout，for database mirroring（联结超时，数据库镜像），715
@@CONNECTIONS function（@@CONNECTIONS函数），298
consistency（一致性），115
constraints（约束），157, 168–176
 dropping from tables（从表中删除），176
 enabling/disabling（启用 / 禁用），173
 forms of（形式），168
 renaming（重命名），687–689
 vs. triggers（与触发器相比较），373
 validating integrity of（验证完整性），674–682
CONTAINS predicate（CONTAINS谓词），8, 230, 232
 for inflectional matches（变形匹配），233
 for term proximity（词的相邻），234
 for wildcard searches（通配符搜索），233
CONTAINSTABLE command（CONTAINSTABLE命令），236
contracts，for message types（约定，消息类型），584
control-of-flow（控制流），307, 312–319
conversations（会话）
 between databases（在数据库之间），580
 dialog（对话），589
 ending（结束），594
 prioritizing（设置优先级），579, 596–598
CONVERT function（CONVERT函数），284–288
Coordinated Universal Time (UTC)（协调世界时），144, 278
COPY_ONLY option（COPY_ONLY选项），804
correlated subqueries（相关的子查询），27
COS function（COS函数），261
COT function（COT函数），262
COUNT aggregate function（COUNT聚合函数），257，259
COUNT_BIG aggregate function（COUNT_BIG聚合函数），250，257

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

covering queries（覆盖查询），209
covering_permission_name（覆盖权限名称），503
@@CPU_BUSY function（@@CPU_BUSY函数），298
CREATE APPLICATION ROLE command（CREATE APPLICATION ROLE命令），497
CREATE ASSEMBLY command（CREATE ASSEMBLY命令），405, 409
CREATE ASYMMETRIC KEY command（CREATE ASYMMETRIC KEY命令），555
CREATE BROKER PRIORITY command（CREATE BROKER PRIORITY命令），579,596–598
CREATE CERTIFICATE command（CREATE CERTIFICATE命令），567
CREATE CONTRACT command（CREATE CONTRACT命令），584
CREATE CREDENTIAL command（CREATE CREDENTIAL命令），528
CREATE DATABASE command（CREATE DATABASE命令），622, 624
 collations and（排序规则），641
 database snapshots and（数据库快照），718
 external access and（外部访问），640
CREATE DATABASE FOR ATTACH command（CREATE DATABASE FOR ATTACH命令），634
CREATE DATABASE AUDIT SPECIFICATION command（CREATE DATABASE AUDIT SPECIFICATION命令），535–539
CREATE ENDPOINT command（CREATE ENDPOINT命令），700
CREATE EVENT NOTIFICATION command（CREATE EVENT NOTIFICATION命令），612
CREATE FULLTEXT CATALOG command（CREATE FULLTEXT CATALOG命令），217
CREATE FULLTEXT INDEX command（CREATE FULLTEXT INDEX命令），219
CREATE FULLTEXT STOPLIST command（CREATE FULLTEXT STOPLIST命令），226, 229
CREATE FUNCTION command（CREATE FUNCTION命令），344, 348
CREATE INDEX command（CREATE INDEX命令），199, 214, 215, 428
CREATE INDEX...DROP_EXISTING command（CREATE INDEX...DROP_EXISTING命令），206
CREATE LOGIN command（CREATE LOGIN命令），476, 492
 capturing（捕获），612
 SQL Server logins and（SQL Server登录名），481
 Windows logins and（Windows登录名），476
CREATE MASTER KEY command（CREATE MASTER KEY命令），551
CREATE MESSAGE TYPE command（CREATE MESSAGE TYPE命令），582
CREATE PARTITION FUNCTION command（CREATE PARTITION FUNCTION命令），180
CREATE PARTITION SCHEME command（CREATE PARTITION SCHEME命令），180
CREATE PROCEDURE command（CREATE PROCEDURE命令），326–331
CREATE QUEUE command（CREATE QUEUE命令），585, 598
CREATE REMOTE SERVICE BINDING command（CREATE REMOTE SERVICE BINDING命令），610
CREATE RESOURCE POOL command（CREATE RESOURCE POOL命令），782
CREATE ROLE command（CREATE ROLE命令），495
CREATE ROUTE command（CREATE ROUTE命令），608
CREATE SCHEMA command（CREATE SCHEMA命令），514
CREATE SERVER AUDIT command（CREATE SERVER AUDIT命令）530–532
CREATE SERVER AUDIT SPECIFICATION command（CREATE SERVER AUDIT SPECIFICATION命令），533–535
CREATE SERVICE command（CREATE SERVICE命令），587
CREATE SPATIAL INDEX command（CREATE SPATIAL INDEX命令），446
CREATE STATISTICS command（CREATE STATISTICS命令），760，762
CREATE SYMMETRIC KEY command（CREATE SYMMETRIC KEY命令），560
CREATE SYNONYM command（CREATE SYNONYM命令），732
CREATE TABLE command（CREATE TABLE命令），26，143
CREATE TRIGGER command（CREATE TRIGGER命令），375
CREATE TYPE command（CREATE TYPE命令），365–367, 371
CREATE USER command（CREATE USER命令），489
CREATE VIEW command（CREATE VIEW命令），240, 247
CREATE WORKLOAD GROUP command（CREATE WORKLOAD GROUP命令），782
CREATE XML SCHEMA COLLECTION command（CREATE XML SCHEMA COLLECTION查询），422
credentials，CREATE CREDENTIAL command and（凭据，CREATE CREDENTIAL命令），528
CROSS APPLY clause（CROSS APPLY子句），35
cross-database chaining（跨数据库链），640
CROSS JOINs（交叉联结），29，31
cross-platform queries（跨平台查询），723
cross-server communication（跨服务器通信），601–611
cross-tab queries via PIVOT operator（通过PIVOT运算符进行交叉表查询），39
CTEs (Comment Table Expressions)（CTE（公共表表达式）），56–62
CUBE clause（CUBE子句），49
(WITH) CUBE clause（(WITH) CUBE clause子句），46
CURRENT_TIMESTAMP function（CURRENT_TIMESTAMP函数），278

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

cursor options，configuring（游标选项，配置），642–643
cursors（游标），307, 319–323
caution against using（使用警告），307, 321
temporary tables as alternative to（作为备选方案的临时表），176

D

data（数据）
grouping（分组），14–17
recovering（恢复），44, 720
relocating（重新分配），657
removing from database（从数据库中删除），656
summarizing（汇总），46–56
unstructured, storing on file system（非结构化的数据在文件系统中存储），80–84
Data Collector（数据收集器），748
data compression（数据压缩），192–195
Data Definition Language (DDL)（数据定义语言），373
data hierarchies（数据层次结构），62
data integrity，constraints and（数据完整性，约束），168, 173
Data Manipulation Language (DML)（数据操作语言），373
data modifications（数据修改），63–90
advanced techniques for（高级技术），89–93
capturing/tracking（捕获/跟踪），93–113
chunking（分块），89
data pages，restoring（数据页，还原），819
data recovery，INTERSECT/EXCEPT operands and（数据恢复，INTERSECT/EXCEPT操作），44
data sources（数据源）
advanced techniques for（高级技术），38–46
querying multiple（多表查询），28–35
data stream compression（数据流压缩）697, 711
data types （数据类型）
columns and（列），144
converting（转换），284–288
data types deprecated with SQL Server 2005（SQL Server 2005不建议使用），76
large value（大型值），76
Database Audit Specification（数据库审核规范），529, 538, 542
creating（创建），535
managing（管理），543–545
removing（移除），545
database certificates（数据库证书），567–573
database design，query performance and（数据库设计，查询性能），739
Database Encryption Key (DEK) （数据库加密密钥），547, 575, 576
Database Engine Tuning Advisor（数据引擎调优顾问），755
database files(数据库文件)
logical name of，changing（逻辑名称，修改），658
managing（管理），654–663
shrinking（收缩），665–668
size of，changing（大小，修改），659
database integrity（数据库完整性），669–686
Database Master Keys（数据库主密钥），547, 550, 551–555
backing up/restoring（备份/还原），553
creating（创建），582
removing Service Master Key encryption from（删除服务主密钥加密），554
database mirroring（数据库镜像），697–716, 801
architecture of（体系结构），698
enabling (steps)（启用（步骤）），700
monitoring/configuring options for（监视/配置选项），714–716
sessions of（会话），707, 710, 713
setting up (steps summarized)（设置（总结的步骤）），709
database objects（数据库对象）
aliases and（别名），732
checking allocation/structural integrity of（检查分配/结构完整性），672–674
identifying user-defined types and（识别用户定义类型），367
maintenance for，（维护）687–690
referenced/referencing, management views for identifying,（被引用/引用，识别管理视图）692
tracking dependencies and（跟踪依赖关系），687, 690–695
user-created，renaming（用户创建的，重命名），687–689
viewing definition of（查看定义），694
database options（数据库选项）
configuring（配置），635–651
viewing（查看），636
database owner，changing（数据库所有者，修改），653
database principals（数据库主体），475, 488–499
database redundancy，database mirroring and（数据库冗余，数据库镜像），697
database roles（数据库角色），488, 493–497
fixed（固定的），493–495
user-defined（用户定义的），495–497
database scope（数据库范围），501
database-scoped events, capturing（数据范围事件，捕获），535–539
database-scoped securables(数据库范围的安全对象)，509–514
database state（数据库状态），651–654
Database Tuning Advisor（数据库优化顾问），203
database user access modes（数据库用户访问模式），628
database users（数据库用户），488–493
creating（创建），489
dropping（删除），491
fixing orphaned（修复孤立用户），491–493
DATABASEPROPERTYEX function（DATABASEPROPERTYEX函数），156, 299
databases （数据库）
access/ownership，controlling(访问/所有权，控制)，651–654
attaching（附加），634–635

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

automatic options for（自动化选项），638–639
backing up/restoring（备份/还原），789–668
detaching（分离），632
enabling for Service Broker（启用Service Broker），581
external access and（外部访问），640
files/filegroups，managing for(文件/文件组，管理)，654–663
I/O statistics and（I/O统计信息），753
maintaining（维护），669–686
partner（伙伴），700
principal（主体），687, 699–716
read-only, setting to（只读，设置为），662
removing assemblies from（从中移除程序集），418
renaming（重命名），631
shrinking（收缩），665–668
snapshots of（快照），717–722
space usage，viewing/managing（空间使用，查看/管理），663–668
viewing information about(查看关于……的信息)，622–624
DATALENGTH function（DATALENGTH函数），264, 269
date conversions（日期转化），285
date functions（日期函数），277–284
DATEADD function（DATEADD函数），280
DATEDIFF function（DATEDIFF函数），278，281
@@DATEFIRST function（@@DATEFIRST函数），293
DATENAME function（DATENAME函数），278, 282
DATEPART function（DATEPART函数），278, 282
datetime data type（datetime数据类型），180, 277, 285
DATE_CORRELATION_OPTIMIZATION option（DATE_CORRELATION_OPTIMIZATION选项），643
DAY function（DAY函数），283
DB lock resource（DB锁资源），124
DBCC CHECKALLOC command（DBCC CHECKALLOC命令），670–672
DBCC CHECKCATALOG command（DBCC CHECKCATALOG命令），681–682
DBCC CHECKCONSTRAINTS command（DBCC CHECKCONSTRAINTS命令），679–681
DBCC CHECKDB command（DBCC CHECKDB命令），669, 672–674
DBCC CHECKFILEGROUP command（DBCC CHECKFILEGROUP命令），675–676
DBCC CHECKIDENT command（DBCC CHECKIDENT命令），166
DBCC CHECKTABLE command（DBCC CHECKTABLE命令），676–679
DBCC commands（DBCC命令），669–682
DBCC FREEPROCCACHE command(DBCC FREEPROCCACHE命令)，342, 645
DBCC INPUTBUFFER command（DBCC INPUTBUFFER命令），134, 321
DBCC OPENTRAN command（DBCC OPENTRAN命令），120
DBCC SHOW_STATISTICS command(DBCC SHOW_STATISTICS命令)，765
DBCC SHRINKDATABASE command（DBCC SHRINKDATABASE命令），665–668
DBCC SHRINKFILE command（DBCC SHRINKFILE命令），665–668
DBCC SQLPERF command（DBCC SQLPERF命令），663
DBCC TRACEOFF command（DBCC TRACEOFF命令），138
DBCC TRACEON command（DBCC TRACEON命令），138, 141
DBCC TRACESTATUS command（DBCC TRACESTATUS命令），138, 140
dbo schemas（dbo架构），514
DB_ID function（DB_ID函数），300
DB_NAME function（DB_NAME函数），300
DDL (Data Definition Language)（DDL（数据定义语言）），373
DDL triggers（DDL触发器），386–393
database-level events and，auditing（数据库级事件，审核），387–388
server-level events and，auditing（服务器级事件，审核），389
viewing metadata and（查看元数据），392
deadlock priority, setting,（死锁优先级，设置）141
deadlocking（死锁），137–142
DEALLOCATE statement（DEALLOCATE语句），321
DECLARE CURSOR statement（DECLARE CURSOR语句），320
DecryptByAsmKey function（DecryptByAsmKey函数），558
DecryptByCert system function（DecryptByCert系统函数），572
DecryptByKey function（DecryptByKey函数），564, 573
DecryptByKeyAutoAsymKey(DecryptByKeyAutoAsymKey函数)，573
DecryptByKeyAutoCert(DecryptByKeyAutoCert函数)，573
DecryptByPassPhrase function(DecryptByPassPhrase函数)，548
DEFAULT constraints（DEFAULT约束），168, 174
default values，inserting rows via（默认值，插入行），65
DEGREES function（DEGREES函数），262
DEK (Database Encryption Key)（数据库加密密钥），547, 575
DELETE statement（DELETE语句），86–88
MERGE command and（MERGE命令），90–93
OUTPUT clause, using with（使用OUTPUT子句），93
DENSE_RANK function（DENSE_RANK函数），288, 292
DENY command（DENY命令），480
database-scoped securables and（数据库范围的安全对象），510
object permissions and（对象权限），520
schema-scoped securables and（架构范围的安全对象），515
securables and（安全对象），501
server-scoped securables and(服务器范围的安全对象)，506
DENY CONNECT SQL command（DENY CONNECT SQL命令），480
derived queries（衍生查询），56–62
derived tables（衍生表），33
dialog conversations（对话会话），589
dialog security（对话安全性），601, 606–608
dictionary compression（目录压缩），193

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

DIFFERENCE function（DIFFERENCE函数），264, 267
differential backups（差异备份），792, 805–809
 restoring（还原），815, 816
 viewing backup metadata and（查看备份元数据），808
DIFFERENTIAL keyword（DIFFERENTIAL关键字），805
dirty reads（脏读），128
DISABLE option（DISABLE选项），127
DISABLE TRIGGER command（DISABLE TRIGGER命令），394
disabling （禁用）
 Change Data Capture（变更数据捕获），107
 change tracking, for full-text indexes（全文索引的修改跟踪），224
 constraints（约束），173
 indexes（索引），205
 query parallelism（并发查询），618
 triggers（触发器），331
disk errors（磁盘错误），650
disk space allocation structures, checking consistency of（磁盘空间分配结构，检查一致性），670–672
disk storage usage, viewing/managing（磁盘存储使用情况，查看/管理），663–668
displaying（显示）. 参见 viewing
DISTINCT keyword（DISTINCT关键字），21
distinct rows（不重复行），44
distributed queries（分布式查询），723–738
distributed partitioned views（分布式分区视图），239, 252
divide, assign (/=) operator（除等于，赋值运算符），84
DLL files, compiling assemblies into（DLL文件，编译程序集），408
DML (Data Manipulation Language)（数据操纵语言），373
DML triggers（DML触发器），374–385
 controlling based on modified columns（根据修改的列控制），384
 transactions and（事务），381
 viewing metadata and（查看元数据），385
DMVs (Dynamic Management Views)（动态管理视图），121, 124,134
downloading AdventureWorks sample database（删除AdventureWorks示例数据库），1
DROP APPLICATION ROLE command（DROP APPLICATION ROLE命令），499
DROP ASYMMETRIC KEY command（DROP ASYMMETRIC KEY命令），560
DROP BROKER PRIORITY command（DROP BROKER PRIORITY命令），597
DROP CERTIFICATE command（DROP CERTIFICATE命令），569
DROP command（DROP命令），154, 599
DROP DATABASE AUDIT SPECIFICATION command（DROP DATABASE AUDIT SPECIFICATION命令），545
DROP DATABASE command（DROP DATABASE命令），632, 720
DROP ENDPOINT command（DROP ENDPOINT命令），714
DROP FULLTEXT CATALOG command（DROP FULLTEXT CATALOG命令），230
DROP FULLTEXT INDEX ON command（DROP FULLTEXT INDEX ON命令），229
DROP FULLTEXT STOPLIST command（DROP FULLTEXT STOPLIST命令），229
DROP INDEX command（DROP INDEX命令），206, 429
DROP LOGIN command（DROP LOGIN命令）
 capturing（捕获），612
 SQL Server logins and（SQL Server登录名），485
 Windows logins and（Windows登录名），479
DROP MASTER KEY command（DROP MASTER KEY命令），552
DROP PROCEDURE command（DROP PROCEDURE命令），332
DROP RESOURCE POOL command（DROP RESOURCE POOL命令），787
DROP ROLE command（DROP ROLE命令），495
DROP SCHEMA command（DROP SCHEMA命令），514
DROP SERVER AUDIT SPECIFICATION command（DROP SERVER AUDIT SPECIFICATION命令），545
DROP STATISTICS command（DROP STATISTICS命令），766
DROP SYMMETRIC KEY command（DROP SYMMETRIC KEY命令），567
DROP SYNONYM command（DROP SYNONYM命令），732
DROP TABLE command（DROP TABLE命令），177
DROP TRIGGER command（DROP TRIGGER命令），399
DROP TYPE command（DROP TYPE命令），368
DROP USER command（DROP USER命令），491
DROP VIEW command（DROP VIEW命令），245
DROP WORKLOAD GROUP command（DROP WORKLOAD GROUP命令），787
DROP XML SCHEMA COLLECTION command（DROP XML SCHEMA COLLECTION命令），422
dropping（删除）
 application roles（应用程序角色），499
 asymmetric keys（非对称密钥），560
 backup devices（备份设备），800
 database certificates（数据库证书），569
 Database Master Keys（数据库主密钥），552
 database roles（数据库角色），495
 database users（数据库用户），491
 databases（数据库），632
 endpoints（端点），714
 error messages（错误消息），462
 indexes（索引），206, 429
 linked server login mappings（链接服务器登录名映射），730
 linked servers（链接服务器），727

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

plan guides（计划指南），772
resource pools（资源池），787
schema-scoped securables（架构范围的安全对象），514
SQL Server logins（SQL Server登录名），485
statistics（统计信息），766
symmetric keys（对称密钥），567
synonyms（同义词），732
triggers（触发器），399
user-defined functions（用户定义函数），356
Windows logins（Windows登录名）479
workload groups（工作负荷组），787
XML Schema collections（XML架构排序规则），422
duplicate rows（重复行），35
duplicate values，removing（重复值，删除），21
durability（持久性），115
Dynamic Management View（动态管理视图），121, 124,

E

EKM (Extensible Key Management)（可扩展的密钥管理），547
emergency state（紧急情况），652
ENABLE TRIGGER command(ENABLE TRIGGER命令)，394
ENABLE_BROKER option（ENABLE_BROKER选项），581
EncryptByAsmKey function（EncryptByAsmKey函数），557
EncryptByCert system function（EncryptByCert系统函数），571
EncryptByKey function（EncryptByKey函数），563
EncryptByPassPhrase function（函数），548
encryption（加密），547–578
certificate（证书），567–573
stored procedures and（存储过程），336
Transparent Data Encryption and（透明数据加密），547, 575–578
ENCRYPTION OPTION, for views（查看ENCRYPTION OPTION），247
END CONVERSATION command（END CONVERSATION命令），590, 594
endpoints（端点）
creating for database mirroring（创建数据库镜像），700–704
removing（移除），714
services defining（服务定义），587
entity integrity（实体完整性），157
equality (=) operator（等于运算符），84
error handling（错误处理），459–473
@@ERROR function（@@ERROR函数），466–468
error messages（错误消息）
dropping（删除），462
invoking with RAISERROR command（调用RAISERROR命令），463
viewing（查看），459
error trapping（捕捉错误），465–468
ESCAPE keyword（ESCAPE关键字），8, 12
event notifications（事件通知），579, 612–614
EXCEPT operator（EXCEPT运算符），44
exclusive locks（排他锁），123
EXECUTE AS statement（EXECUTE AS语句），340–337, 524
EXECUTE command（EXECUTE命令），767
execution runtime information，viewing（执行运行时信息，查看），746–748
EXISTS operator（EXISTS运算符），8
EXP function（EXP函数），262
EXPAND VIEWS hint（EXPAND VIEWS提示），452
explicit transactions（显式事务），116–119
expressions（表达式），7–14
extended stored procedures（扩展存储过程），403
Extensible Key Management (EKM)（可扩展密钥管理），547
extent lock resource（扩展锁资源），124
extents（区），758
EXTERNAL_ACCESS permission（EXTERNAL_ACCESS权限），406, 409, 414

F

failover clustering（故障转移集群），697
failovers（故障转移），709, 710, 712
FAST integer hint（FAST整型提示），451
FASTFIRSTROW hint（FASTFIRSTROW提示），455
FILE keyword（FILE关键字），805, 816
file lock resource（文件锁资源），124
FILEGROUP keyword（FILEGROUP关键字），805, 816
filegroups（文件组），180–184, 187, 627
adding to databases（添加到数据库），660
backing up（备份），805–807
default，setting，（默认，设置）660
indexes，creating on，（索引，创建）212
managing（管理），654–663
placing tables on（将表放入），180, 191
read-only（只读），662, 807
removing（删除），661
files, I/O statistics and（文件，I/O统计信息），753
files，backing up（文件，备份），805–807
FILESTREAM attribute（FILESTREAM属性），80–84
FILLFACTOR option（FILLFACTOR选项），209
filtered indexes（筛选索引），212, 214
filtered statistics（筛选统计信息），762
fixed server roles（固定服务器角色），485–488
FLOOR function（FLOOR函数），262
fn_my_permissions function（fn_my_permissions函数），523
FOR XML command（FOR XML命令），430–433
FORCESEEK hint（FORCESEEK提示），456–458
foreign key constraints（外键约束），158, 168
foreign keys（外键），88, 160–164
FORMSOF command（FORMSOF命令），234
four-part linked server names（4部分组成的链接服务器名称），730, 732
FREETEXT predicate（FREETEXT谓词），8, 230, 235
FREETEXTTABLE ranking function（FREETEXTTABLE排名函数），235
FROM clause（FROM语句），235
best practices for（最佳实践），741

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

distributed queries and（分布式查询），730, 733
predicates and（谓词），230
rows，updating via（行，更新），75
views and（视图），239
full backups（完整备份），791, 793–804
COPY_ONLY option and（COPY_ONLY选项），804
restoring（还原），810–812
FULL OUTER JOINs，29
FULLTEXTCATALOGPROPERTY system function（FULLTEXTCATALOGPROPERTY系统函数），225
FULL recovery model（FULL恢复模式），649, 700, 705, 792
FULL safety mode（FULL安全模式），711
full table scan（全表扫描），197
Full-Text Engine for SQL Server (MSFTESQL)（SQL Server全文引擎），217
full-text catalogs（全文目录），217–230
full-text indexes（全文索引），217–230
full-text search（全文搜索），217–261
advanced（高级的），232–232
ranked（排名的），235–237
CLR scalar UDFs, creating（创建CLR标量UDF），412–414
date（日期），277–284
flexible, using ISNULL（灵活的使用ISNULL），275
mathematical（数学），261
ranking（排名），288–293
statistical（统计），298
string（字符串），263–303
user-defined（用户定义），343–365, 372, 404
functions（函数），257–306
aggregate（聚合），257–261
date（日期），277–284
mathematical（数学），261–263
identity（标识），303–306
NULLs and, 274–277
ranking（排名），288–293
string（字符串），262–274
system（系统），293–303
type conversion and（类型转换），284–288
uniqueidentifier（唯一标识符），303–306

G

geography data type（geography数据类型），442–448
Geography Markup Language (GML)（地理标记语言），442
geometry data type（geometry数据类型），442–448
GetAncestor method（GetAncestor方法），438
GETDATE function（GETDATE函数），278, 286
GetDescendant method（GetDescendant方法），437, 439
GetLevel method（GetLevel方法），436, 440
GetReparentedValue method（GetReparentedValue方法），441
GetRoot method（GetRoot方法），440
GETUTCDATE function（GETUTCDATE函数），278
GET_FILESTREAM_TRANSACTION_CONTEXT() method（GET_FILESTREAM_TRANSACTION_CONTEXT()方法），84
global temporary tables（全局临时表），176
GML (Geography Markup Language)（地理标记语言），442
GMT (Greenwich Mean Time)（格林威治标准时间），144, 278
GOTO command（GOTO命令），307, 312, 316
GRANT command（GRANT命令），480
database-scoped securables and（数据库范围的安全对象），509
object permissions and（对象权限），519
schema-scoped securables and（架构范围的安全对象），515
securables and（安全对象），501
server-scoped securables and（服务器范围的安全对象），505
grantee，permissions and（被授权者，权限），505
grantor，permissions and（授予者，权限），505
Greenwich Mean Time (GMT)（格林威治标准时间），144, 278
GROUP BY ALL clause（GROUP BY ALL子句），15
GROUP BY clause（GROUP BY子句），14, 49, 51
GROUP BY ROLLUP clause（GROUP BY ROLLUP子句），48
group-level identification, for rows（行的分组级别标识），53–56
GROUPING aggregate function（GROUPING聚集函数），257
grouping data（分组数据），14–17
GROUPING function（GROUPING函数），51, 54
GROUPING SETS operator（GROUPING SETS运算符），49
GROUPING_ID function（GROUPING_ID函数），51, 53–56

H

Hardware Security Modules (HSM)（硬件安全模块），547
hardware，query performance and（硬件，查询性能），740
HASH joins（HASH联结），450
{HASH | ORDER} GROUP hint（{HASH | ORDER} GROUP提示），451
Has_perms_by_name system function（Has_perms_by_name系统函数），522
HAVING command（HAVING命令），16,230
heaps（堆），197, 669, 686
hierarchyid data type（层次结构数据类型），62, 435–441
high availability（高可用性），697, 710
high performance（高性能），711
high protection（高保护），711
hints（提示），449–458, 771–775
HONOR_BROKER_PRIORITY option（HONOR_BROKER_PRIORITY选项），596
horizontal partitioning（水平分区），180
horizontally partitioned tables（水平分区表），239, 251, 256
HOST_ID function（HOST_ID函数），301
HOST_NAME function（HOST_NAME函数），301
hot database standby（热备用数据库），698
HSM (Hardware Security Modules)（硬件安全模块），547

I

I/O contention, identifying（标识出I/O争用），753

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

IDENTITY column property（IDENTITY列属性），66，165，303–306
IDENTITY values，checking（IDENTITY值，检查），166
IDENT_CURRENT function（IDENT_CURRENT函数），304
IDENT_INCR function（IDENT_INCR函数），305
IDENT_SEED function（IDENT_SEED函数），305
@@IDLE function（@@IDLE函数），298
IF...ELSE statement（IF...ELSE语句），307，310
IGNORE_CONSTRAINTS hint（IGNORE_CONSTRAINTS提示），455
IGNORE_TRIGGERS hint（IGNORE_TRIGGERS提示），455
image data types (deprecated with SQL Server 2005)（image数据类型（SQL Server 2005 不建议使用）），76
image files，inserting/updating via OPENROWSET and BULK（image文件，通过OPTNROWSET和BULK插入/更新），78
implicit transactions（隐式事务），116
IN operator（IN运算符），8
INCLUDE command（INCLUDE命令），199，209
@@IDENTITY function（@@IDENTITY函数），304
INDEX (index_val [,... n]) hint（INDEX (index_val [,... n])提示），455
index columns，sort direction for（索引列，排序方向），203
index compression（索引压缩），212
index fragmentation（索引碎片），755，756–758
index key（索引键），197，199
index key columns（索引键列），197
index pages（索引页），210
index seeks/index scans, FORCESEEK hint and（索引查找/索引扫描，FORCESEEK提示），456–458
index usage，displaying（显示索引使用），754，759–760
indexed views（索引视图），239，246，248–251
indexes（索引），197–215
 access to，allowing during creation（在创建时允许访问），208
 best practices for（最佳实践），755
 clustered（聚集索引），198，755
 creating（创建），199–230
 maintenance for（维护），682–686
 managing（管理），212–215
 multiple-column，creating（创建多列），202
 nonclustered（非聚集索引），198，755
 page-level data compression for（页级别数据压缩），684
 performance tuning for（性能调优），207–209，212，754–760
 primary key and（主键），157
 query performance and（查询性能），199，739
 re-creating after dropping（删除后重新创建），206
 renaming（重命名），687–689
inflectional matches（变形匹配），217，233
INFORMATION_SCHEMA.columns system（INFORMATION_SCHEMA列系统），24
inline user-defined functions（内联用户定义函数），343，349
INNER JOINs，29
INSERT statement（INSERT语句），63–71
 MERGE command and（MERGE命令），90–93
 OUTPUT clause，using with（OUTPUT子句，使用），93
INSERT...EXEC statement（INSERT...EXEC语句），70
INSERT...SELECT statement（INSERT...SELECT语句），68
INSTEAD OF DML triggers（INSTEAD OF DML触发器），378
INSTEAD OF triggers（INSTEAD OF触发器），247，374
integers，FAST integer hint for（整型，FAST整型提示），451
intent locks（意向锁），123
intermediate indexes，tempdb system database and（临时索引，tempdb系统数据库），207
intermediate leaf level（中间叶级别），198
intermediate level pages（中间级别页），210
international data storage，collations and（国际化数据存储，排序规则），155
INTERSECT operand,（INTERSECT运算符），44
INTO clause（INTO子句），26
@@IO_BUSY function（@@IO_BUSY函数），298
IS NOT NULL operator（IS NOT NULL运算符），8
IS NULL operator（IS NULL运算符），10
ISDATE function（ISDATE函数），287
IsDescendantOf method（IsDescendantOf方法），441
ISNULL function（ISNULL函数），274
ISNUMERIC function（ISNUMERIC函数），287
isolation（隔离），115，128–133

J

join hints（联结提示），449
JOIN keyword（JOIN关键字），28
Joins,best practices for（联结，最佳实践），741

K

KEEP PLAN hint（KEEP PLAN提示），452
KEEPDEFAULTS hint（KEEPDEFAULTS提示），455
KEEPFIXED PLAN hint（KEEPFIXED PLAN提示），452
KEEPIDENTITY hint（KEEPIDENTITY提示），455
key columns（键列），199
key lock resource（键锁资源），124
key-range locks（键范围锁），123
keys（键），157–168，359
 multiple（多个），199
 surrogate（代理），165–168，359
KILL command（KILL命令），134

L

@@LANGID function（@@LANGID函数），294
@@LANGUAGE function（@@LANGUAGE函数），294
large value data types（大值数据类型），76
leaf level（叶级别），198
leaf level pages（叶级别页），210
LEFT function（LEFT函数），264，268
LEFT OUTER JOINs，29
LEN function（LEN函数），264，269
LIKE operator（LIKE运算符），8
 wildcards，using with（使用通配符），11

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

LIKE operator, vs. full-text search（LIKE运算符与全文搜索比较），217, 230
linked servers（链接服务器），723–738
lists，comma-delimited （逗号分隔的列表），25
local partitioned tables（局部分区表），256
local temporary tables（局部临时表），176
local variables，xml data types for（局部变量，xml数据类型），419
location applications, geography/geometry data types and（定位应用程序，地理/几何数据类型），442
lock escalation（锁升级），124, 126
lock mode（锁定模式），123
@@LOCK_TIMEOUT function（@@LOCK_TIMEOUT函数），295
locking（锁定），122–133
 indexes and（索引），211
 viewing lock activity and（查看锁活动），124
locks（锁），617
LOG function（LOG函数），262
log sequence number (LSN) （日志序列号），821
log shipping（日志传送），698
LOG10 function（LOG10函数），262
logical names，for backup devices（逻辑名称，备份设备），799
logical operators（逻辑运算符），743
login commands，capturing（登录名命令，捕捉），612–614
login/user associations（登录名/用户的关联），491–493
LOGINPROPERTY function（LOGINPROPERTY函数），484
logins（登录名）
 SQL Server，480–485
 Windows，475–480
logon triggers（登录触发器），390
lookup tables，temporary tables and（查找表，临时表），176
lookups，multiple（多重查找），177
LOOP joins（LOOP联结），450
{LOOP | MERGE | HASH} JOIN hint（{LOOP | MERGE | HASH} JOIN提示），451
lost updates（更新丢失），128
LOWER function（LOWER函数），264, 270
LSN (log sequence number) （日志序列号），821
LSN boundaries（LSN边界），105
LTRIM function（LTRIM函数），264, 271

M

mapping applications, geography/geometry data types and（地图应用程序，地理/几何数据类型），442
master database（主数据库），581, 603
Master Keys（主密钥），547, 550–555
master merges（主合并），221
matching rows（行匹配），27, 44
mathematical functions（数学函数），261
MAX aggregate function（MAX聚集索引），257, 259
MAXDOP index option（MAXDOP索引选项），208
MAXDOP number_of_processors hint（MAXDOP处理器数据提示），451
MAXRECURSION hint（MAXRECURSION数据提示），61, 452
media families（媒体簇），799
media sets（媒体集），797, 799, 808
MERGE command（MERGE命令），89, 90–93
MERGE joins（MERGE联结），450
MERGE RANGE functionality（MERGE RANGE功能），186, 188
message types（消息类型），582
 message-queuing，Service Broker for（消息队列，Service Broker），579–601
 receiving/responding to（接收/响应），591
 sending（发送），589
metadata lock resource（元数据锁资源），124
Microsoft
 Full-Text Engine for SQL Server (MSFTESQL)（SQL Server全文引擎），217
 .NET framework（.NET框架），402
 SQL Native Client OLE DB Provider，724, 734
MIN aggregate function（MIN聚合函数），257, 259
MIRROR TO clause（MIRROR TO语句），801
mirroring backup sets（镜像备份集），801
mirroring database See database mirroring（镜像数据库，参见数据库镜像）
model system database（模型系统数据库），622
MODIFY FILE command（MODIFY FILE命令），657–659
MODIFY FILEGROUP command（MODIFY FILEGROUP命令），660，662
modifying data. *See* data modifications（修改数据，参见数据修改）
modulo, assign (%=) operator（模等于，赋值运算符），84
MONTH function（MONTH函数），278, 283
msdb.dbo.suspect_pages system table（msdb.dbo.suspect_pages系统表），819
MSFTESQL (Full-Text Engine for SQL Server)（SQL Server全文引擎），217
multi-statement user-defined functions（多语句用户定义函数），343, 351,362–365
multilingual data storage，collations and（多语言数据存储，排序规则），155
multiple keys（多键），199
multiple-column indexes（多列索引），202
multiply, assign (*=) operator（乘等于，赋值运算符），84
MULTI_USER mode（MULTI_USER模式），628

N

names，displaying in single column（名称，在单个列中显示），25
NCHAR function（NCHAR函数），264
nesting TRY...CATCH calls（嵌套调用TRY...CATCH），471
@@NESTLEVEL function（@@NESTLEVEL函数），295

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

.NET assemblies（.NET程序集），401
.NET CLR assemblies（.NET CLR程序集），325
.NET framework（.NET框架），402
network throughput，query performance and（网络吞吐量，查询性能），740
NEWID system function（NEWID系统函数），303, 305
NEWSEQUENTIALID system function（NEWSEQUENTIALID系统函数），303, 306
NOCHECK option（NOCHECK选项），172
NOEXPAND keyword（NOEXPAND关键字），251
noise words（干扰词），217, 226–229
NOLOCK hint（NOLOCK提示），455
non-key columns（非键列）
 adding to nonclustered index via INCLUDE command（增加非聚集索引与INCLUDE命令相比较），209
 enforcing uniqueness for（强制唯一），201
 including within a nonclustered index（包含非聚集索引），209
non-recursive CTEs（非递归CTE），56
non-unique indexes（非唯一索引），199
nonclustered indexes（非聚集索引），198
 best practices for（最佳实践），755
 compression and（压缩），215
 non-key columns and（非键列），209
 partitioning applied to（分区应用到），213
nonrepeatable reads（不可重复读），128
NORECOVERY option，database mirroring and（NORECOVERY选项，数据库镜像），705
NOT BETWEEN operator（NOT BETWEEN运算符），8
NOT IN operator（NOT IN运算符），8
NOT LIKE operator（NOT LIKE运算符），8
ntext data types (deprecated with SQL Server 2005)(ntext数据类型，(SQL Server 2005不建议使用))，76
NTILE function（NTILE函数），288, 292
NULL values（NULL值），274–277
 checking for（检查），10
 sparse columns and（稀疏列），150–153
NULL | NOT NULL option（NULL | NOT NULL选项），144
nullability（是否可为null），144
NULLIF function（NULLIF函数），274, 277
numeric data types（numeric数据类型），146
nvarchar(max) data types（nvarchar(max)数据类型），76

O

object dependencies（对象依赖关系），244
 identifying（标识），690–692
 tracking（跟踪），687, 690–695
object lock resource（对象锁资源），124
object permissions（对象权限），519–522
objects（对象）
 changing schemas for（修改架构），689
 reporting information about（报告信息），154
OBJECT_DEFINITION function（OBJECT_DEFINITION函数），243, 694
OBJECT_ID function（OBJECT_ID函数），301, 694
OBJECT_NAME function(OBJECT_NAME函数)，125, 301
offline state（离线状态），652
OGC (Open Geospatial Consortium)（开放地理空间联盟），442
OLE DB providers，723, 730, 734
ON DELETE command（ON DELETE命令），163
ON SCHEMA command（ON SCHEMA命令），519
ON UPDATE command（ON UPDATE命令），163
ONLINE option（ONLINE选项），208
online restores（在线还原），810
online state（在线状态），652
Open Geospatial Consortium (OGC)（开放地理空间联盟），442
OPEN SYMMETRIC KEY command（OPEN SYMMETRIC KEY命令），562, 573
OPENQUERY command（OPENQUERY命令），730, 733
OPENROWSET command（OPENROWSET命令），78, 81, 83, 723, 733–738
OPENXML command（OPENXML命令），430, 433–435
operators（运算符），7–14
OPTIMIZE FOR (@variable_name = literal_constant) [, ...n] hint（OPTIMIZE FOR (@variable_name = literal_constant) [, ...n]提示），452
ORDER BY clause（ORDER BY子句）
 syntax for（语法），17–19
 best practices for（最佳实践），741
OUTER APPLY clause（OUTER APPLY子句），37
OUTER JOINs，29
OUTPUT clause（OUTPUT子句），93
OUTPUT parameters（OUTPUT参数），328, 330
overlapping rows（重叠行），46
ownership chaining（所有权链），337

P

@@PACKET_ERRORS function（@@PACKET_ERRORS函数），298
@@PACK_RECEIVED function（@@PACK_RECEIVED函数），298
@@PACK_SENT function（@@PACK_SENT函数），298
PAD_INDEX option（PAD_INDEX选项），209
page compression（页压缩），192–195
PAGE keyword（PAGE关键字），819
page-level data compression, for indexes（索引的页级数据压缩），684
page lock resource（页锁资源），124
page locks（页锁），211
page splits（页拆分），210
page verification，configuring（页验证，配置），650
parallelism，index build performance and（并行性，索引创建性能），208
parameterization（参数化），459, 778–781
PARAMETERIZATION option（PARAMETERIZATION选项），644

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

PARAMETERIZATION { SIMPLE | FORCED } hint（PARAMETERIZATION { SIMPLE | FORCED }提示），452
parameterized stored procedures（参数化存储过程），328
parameters（参数），328
 user-defined types and，identifying（标识用户定义类型），367
 xml data types for（xml数据类型），419
parent_class_desc，503
parent_covering_permission_name，503
Parse method（Parse方法），439, 445
partial backups（部分备份），807
PARTIAL command（PARTIAL命令），818
partial restores（部分还原），818
partition functions，dropping（分区函数，删除），190
partition-level compression（分区级别压缩），194
partition schemes（分区架构），181, *190*
partitioned views（分区视图），251–256
partitions（分区），197
 mapping columns to（映射列），180
 removing（移除），188
partner databases（伙伴数据库），700
passphrases，encryption and（通行短语，加密），547–550
passwords，encryption and（密码，加密），*547–550*
PATH indexes（PATH索引），429
PATINDEX function（PATINDEX函数），263, 266, 269
Performance（性能）
 Change Data Capture (CDC)（变更数据捕获）and, 93, 96
 cursors and（游标），307, 319
 data compression and（数据压缩），194
 filegroups and（文件组），180
 for indexes, improving（改进索引），207–209, 212
 index build, parallelism and（索引创建，并行度），208
 indexed views and（索引视图），248
 optimizing. *See* performance tuning（优化，参见性能调优）
 stored procedures and（存储过程），340
 temporary tables/table variables and（临时表/表变量），177
 transactions and（事务），129
 triggers and（触发器），373
 user-defined functions and（用户定义函数），357
 views and（视图），240
 XML indexes and（XML索引），428
performance tuning（性能调优），739–787
 best practices for（最佳实践），741
 for indexes（索引），754–760
permissions（权限），501–529
 for assemblies，modifying（修改程序集），417
 on database-scoped securables（数据库范围安全对象），509–514
 managing across securable scopes（跨安全对象范围进行管理），522–529
 object（对象），519–522
 principals and（主体），475
 on schema-scoped securables（架构范围安全对象），514–519
 on server-scoped securables（服务器范围安全对象），505–509
 viewing available（查看可用的），503
PERSISTED keyword（PERSISTED关键字），149
phantom reads（幻读），128
PI function（PI函数），*262*
PIVOT operator（PIVOT运算符），39
plan guides（计划指南），771–781
 cautions for（警告），771, 773
 checking validity of（验证有效性），777
 creating based on existing query plans（基于即有查询计划创建），775–777
 forcing parameterization and（强制参数化），778–781
point-in-time recovery（时点恢复），649, 814, 820
POWER function（POWER函数），262
predicates（谓词），230–232
prefix compression（前缀压缩），193
price points，DISTINCT keyword for（价格点，DISTINCT关键字），22
primary databases（主数据库）。参见 principal databases，database mirroring and primary filegroup，627
primary key constraints（主键约束），159, 169
primary keys（主键），157–159, 201
principal databases，database mirroring and（主体数据库，数据库镜像），697,699–716
 architecture of（体系结构），698–700
 backing up/restoring（备份/还原），705–707
principals（主体），475–499
private keys（私钥）
 changing password and（修改密码），557
 database certificates and（数据库证书），570
procedure cache，flushing（清空过程缓存），342
processes，controlling number of used in parallelism（处理器，控制并行使用的数量），208
PROPERTY indexes（PROPERTY索引），429
proximity of terms（词的相邻），217, 234

Q

query concurrency，index locking and（查询并发性，索引锁定），211
query execution plans（查询执行计划），327, 340
 forcing SQL Server to use（强制SQL Server使用），769
 viewing（查看），743–771
query hints（查询提示），451
query optimization（查询优化），739–787
 best practices for（最佳实践），741
 hints and（提示），449–787
query parallelism，disabling（禁用查询并行性），618
query performance, indexes and（索引的查询性能），199
querying（查询）
 BULK option and（BULK选项），735–738
 distributed queries and（分布式查询），723–738
 HAVING clause and（HAVING子句），16
 for incoming messages（传入的消息），591
 multiple data sources（多数据源），28–35

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

WHERE clause and（WHERE子句），3–7
queues（队列），580, 585, 752
quorum，of connected servers（仲裁，相连接的服务器），699, 710
QUOTENAME function（QUOTENAME函数），264

R

RADIANS function（RADIANS函数），262
RAISERROR command（RAISERROR命令），460, 462–465
RAND function（RAND函数），262
range queries，clustered indexes and（范围查询，聚集索引），198
RANK function（RANK函数），288, 290
ranking functions（排名函数），288–293
READ COMMITTED isolation level（READ COMMITTED隔离级别），128
read-consistency（读一致性），647–648
READ UNCOMMITTED isolation level（READ UNCOMMITTED隔离级别），128
READPAST hint（READPAST提示），455
READ_COMMITTED_SNAPSHOT database option（READ_COMMITTED_SNAPSHOT数据库选项），647–648
READ_WRITE_FILEGROUPS option（READ_WRITE_FILEGROUPS选项），807
rebuilding heaps（堆重建），669, 686
RECEIVE command（RECEIVE命令）
 messages and（消息），591
 queues and（队列），585
RECEIVE TOP command（RECEIVE TOP命令），594
RECOMPILE hint（RECOMPILE提示），452
recompiling stored procedures（重编译存储过程），340
RECONFIGURE command（RECONFIGURE命令），617, 619
RECONFIGURE WITH OVERRIDE command（RECONFIGURE WITH OVERRIDE命令），396，617
recovery paths（恢复路径），820
recovery plan, for databases（数据库的恢复路径），789–791
recovery point objective (RPO)（恢复点目标），789
recovery time objective (RTO)（恢复时间目标），789
recursive CTEs（递归CTE），56, 59
recursive foreign key references（递归外键引用），162
recursive members（递归成员），61
referenced/referencing objects, identifying（识别被引用/引用的对象），692
relational data sets，converting to XML format（相关的数据集，转化为XML格式），430–435
remote data sources（远程数据源），723
REMOTE joins（REMOTE联结），450
remote service bindings（远程服务绑定），601, 608
REMOVE FILE command（REMOVE FILE命令），656, 662
REMOVE FILEGROUP command（REMOVE FILEGROUP命令），661
REMOVE PRIVATE KEY command（REMOVE PRIVATE KEY命令），571
REPAIR options，DBCC commands and（REPAIR选项，DBCC命令），669, 674
REPEATABLE READ COMMITTED isolation level（REPEATABLE READ COMMITTED隔离级别），129
REPLACE function（REPLACE函数），264, 269
REPLICATE function（REPLICATE函数），264, 272
replication（复制），698
resource consumption, constraining（限制资源争用），781–787
Resource Governor feature（资源调配器特性），781–787
resource pools（资源池），781–787
resources for further reading（更深入地读取资源）
 geography/geometry data types（地理/几何数据类型），445
 Waits and Queues methodology（查询和队列的方法学），752
 XML Schema definitions（XML架构定义），442
 XML syntax（XML语法），442
 XPath，432
 XQuery，425
RESTORE command（RESTORE命令），792, 799
 differential backup and（差异备份），816
 full backup and（完整备份），810
 PARTIAL command and（PARTIAL命令），818
RESTORE FILELISTONLY command（RESTORE FILELISTONLY命令），808
RESTORE...FROM DATABASE_SNAPSHOT command（RESTORE...FROM DATABASE_SNAPSHOT命令），720–722
RESTORE HEADERONLY command（RESTORE HEADERONLY命令），808
RESTORE LABELONLY command（RESTORE LABELONLY命令），808
RESTORE LOG command（RESTORE LOG命令），813
RESTORE MASTER KEY command（RESTORE MASTER KEY命令），553
RESTORE...MOVE command（RESTORE...MOVE命令），812
RESTORE VERIFYONLY command（RESTORE VERIFYONLY命令），808
restoring databases（还原数据库），810–822
 from differential backups（从差异备份中还原），815
 from full backups（从完整备份中还原），810–812
 multiple recovery paths and（多个恢复路径），820
 partial restores and（部分还原），818–819
 from transaction log backups（从事务日志备份中还原），812–815
RETURN command（RETURN命令），307, 312
RETURNS keyword（RETURNS关键字），352
REVERSE function（REVERSE函数），264, 273
REVERT command（REVERT命令），524
REVOKE command（REVOKE命令）
 database-scoped securables and（数据库范围的安全对象），510
 object permissions and（对象权限），520
 schema-scoped securables and（架构范围的安全对象），516
 securables and（安全对象），501
 server-scoped securables and（服务器范围的安全对象），506

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

RID lock resource（RID锁资源），124
RIGHT function（RIGHT函数），264, 268
RIGHT OUTER JOINs，29
ROBUST PLAN hint（ROBUST PLAN提示），452
ROLLBACK command，triggers and（ROLLBACK命令，触发器），383
ROLLBACK TRANSACTION command（ROLLBACK TRANSACTION命令），116
ROLLUP clause（ROLLUP子句），48, 49
root nodes（根节点），198
ROUND function（ROUND函数），262
routes，Service Broker and（路由，Service Broker），601, 608
row-based security（基于行的安全性），365
row compression（行压缩），192–195
row locks（行锁），211
@@ROWCOUNT function（@@ROWCOUNT函数），297
ROWGUIDCOL property（ROWGUIDCOL属性），165, 168, 303
rows（行），35
converted from columns via UNPIVOT（通过UNPIVOT从列转化为），42
count of（计数），259
cursors and（游标），319–323
deleting（删除），86–88
distinct（不重复），44
extracting sampling of via TABLESAMPLE（通过TABLESAMPLE提取样本），38
grouping（分组），14–17
inserting into tables（插入表），64–71
limiting percentage of（限制百分比），20
matching（匹配），27, 44
multiple, inserting via single INSERT（多个插入与单个INSERT相比较），71
overlapping（重叠），46
ranking functions and（排名函数），288–293
returning based on list of values（基于列表值返回），11
revealing rows generated via GROUPING（展现GROUPING生成的行），51
selecting columns for（选择列），2
table-valued functions，invoking for（调用表值函数），35–38
updating（更新），74–85
ROW_NUMBER function（ROW_NUMBER函数），45, 288
RPO (recovery point objective)（恢复点目标），789
RTO (recovery time objective)（恢复时间目标），789
RTRIM function，（RTRIM函数）264, 271

S

SAFE permission（SAFE权限），406, 409, 414
SAVE TRANSACTION command（SAVE TRANSACTION命令），116
scalar user-defined functions（标量用户定义函数），343
advantages of（益处），357
cross-referencing natural key values and（交差引用自然键值），359–362
reusable code and（重用代码），357
scanning table data（扫描表数据），197
schema modification locks（架构修改锁），123
schema scope（架构范围），501
schema-scoped securables（架构范围的安全对象），514–519
schema stability locks（架构稳定锁），123
schema.object_name, 516
SCHEMABINDING option（SCHEMABINDING选项），248
schemas（架构），3
changing for objects（为对象修改），689
validating XML data via（确认XML数据），422–424
scope qualifier（范围限定符），519
SCOPE_IDENTITY function（SCOPE_IDENTITY函数），304
scripts，creating via SELECT statement（通过SELECT语句创建脚本），23
search conditions（搜索条件），6, 75
searches（搜索），参见full-text search
securables（安全对象），475, 501–529
auditing SQL instance-/database-scoped activity against（审核SQL实例/数据库范围的活动），529–545
database-scoped（数据库范围），509–514
managing permissions across（管理权限），522–529
schema-scoped（架构范围），514–519
server-scoped（服务器范围），505–509
security（安全）
dialog（对话），601, 606–608
EXECUTE AS statement and（EXECUTE AS语句），337–340
principals and（主体），475–499
row-based（基于行），365
stored procedures and（存储过程），326，335
Transparent Data Encryption (TDE) for（透明数据加密），547,575
transport（传送），601, 603–606
security-identifiers (sids)（安全标识号，sid），491
SELECT statement（SELECT语句），1–62
best practices for（最佳实践），741
indexed views and（索引视图），248
temporary tables and（临时表），176
temporary tables as alternative to（作为备选方案的临时表），177
VALUES keyword and（VALUES关键字），72
views and（视图），239，241
selectivity，indexes and（索引的选择性），202
semicolon (;)（分号;），59
SEND ON CONVERSATION command（SEND ON CONVERSATION命令），590, 593
SERIALIZABLE isolation level（SERIALIZABLE隔离级别），129
Server Audit object（服务器审核对象），501, 529–545
creating（创建），530–532
enabling（启用），539
modifying（修改），543–545
Server Audit Specification（服务器审核规范），529, 533, 542
creating（创建），535

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

managing（管理），543–545
removing（移除），545
server roles，fixed（服务器角色，固定的），485–488
server scope（服务器范围），501
server-scoped securables（服务器范围安全对象），505–509
server session id，renamed from server process id（服务器会话id，从服务器进程id重命名），321
@@SERVERNAME function（@@SERVERNAME函数），296
SERVERPROPERTY function（SERVERPROPERTY函数），155, 300
servers，linked（链接服务器），723–738
Service Broker，579–133
cross-server communication and（跨服务器通信），601–611
steps in creating applications (list)（创建应用程序（列表）的步骤），580
severity levels, for error messages（错误消息的严重级别），460, 465
shared locks（共享锁），123
SHOWPLAN commands（SHOWPLAN命令），743–746
shrinking databases/database files（收缩数据库/数据库文件），665–668
SHUTDOWN command（SHUTDOWN命令），505
sids (security-identifiers)（安全标识号），491
SIGN function（SIGN函数），262
SIMPLE recovery model（SIMPLE恢复模式），649, 791
SIN function（SIN函数），262
SINGLE_USER mode（SINGLE_USER模式），628
smalldatetime data type（smalldatetime数据类型），277, 285
SNAPSHOT COMMITTED isolation level（SNAPSHOT COMMITTED隔离级别），129
Snapshot Isolation（快照隔离级别），108, 113
snapshots，of databases（快照，数据库），717–722
SOME operator（SOME运算符），8
sort direction for index columns（索引列的排序方向），203
sort order，collations for（排列顺序，排序规则），155
SORT_IN_TEMPDB command（SORT_IN_TEMPDB命令），207
SOUNDEX function（SOUNDEX函数），264, 267
source control systems, benefits of（资源控制文件，益处），335
SPACE function（SPACE函数），264, 272
Sparse columns（稀疏列），150–153
sparse files（稀疏文件），717
spatial data（空间数据）442–448
spatial indexes（空间索引），446
SPID（服务器进程ID），321
@@SPID function（@@SPID函数），296
sp_addextendedproc system-stored procedure（sp_addextendedproc系统存储过程），403
sp_addlinkedserver stored procedure（sp_addlinkedserver存储过程），724, 728
sp_addlinkedsrvlogin stored procedure（sp_addlinkedsrvlogin存储过程），728
sp_addmessage system-stored procedure（sp_addmessage系统存储过程），460
sp_addrolemember system-stored procedure（sp_addrolemember系统存储过程），494
sp_addsrvrolemember system-stored procedure（sp_addsrvrolemember系统存储过程），485
sp_addumpdevice system-stored procedure（sp_addumpdevice系统存储过程），799
sp_cdc_enable_db stored procedure（sp_cdc_enable_db存储过程），99
sp_changedbowner system-stored procedure（sp_changedbowner系统存储过程），651,653
sp_changeobjectowner system-stored procedure（sp_changeobjectowner系统存储过程），689
sp_change_users_login system-stored procedure（sp_change_users_login系统存储过程，不推荐使用），493
sp_configure stored procedure（sp_configure存储过程），396, 404
sp_configure system-stored procedure（sp_configure系统存储过程），615, 618
sp_control_plan_guide system-stored procedure（sp_control_plan_guide系统存储过程），772
sp_createstats stored procedure（sp_createstats存储过程），763–764
sp_create_plan_guide system-stored procedure（sp_create_plan_guide系统存储过程），771, 779
sp_create_plan_guide_from_handle system-stored procedure（sp_create_plan_guide_from_handle系统存储过程），775
sp_depends system-stored procedure（sp_depends系统存储过程，不推荐），690
sp_detach_db system-stored procedure（sp_detach_db系统存储过程），632
sp_dropdevice system-stored procedure（sp_dropdevice系统存储过程），800
sp_droplinkedsrvlogin system-stored procedure（sp_droplinkedsrvlogin系统存储过程），730
sp_droprolemember system-stored procedure（sp_droprolemember系统存储过程），494
sp_dropserver system-stored procedure（sp_dropserver系统存储过程），727
sp_dropsrvrolemember stored procedure（sp_dropsrvrolemember存储过程），486
sp_estimate_data_compression_savings system stored procedure（sp_estimate_data_compression_savings系统存储过程），193
sp_executesql system-stored procedure（sp_executesql系统存储过程），740, 766,767, 773
sp_get_query_template system-stored procedure（sp_get_query_template系统存储过程），779
sp_grantdbaccess system-stored procedure（sp_grantdbaccess系统存储过程），489
sp_help stored procedure（sp_help存储过程），154
sp_helpdb stored procedure（sp_helpdb存储过程），622
sp_helpdbfixedrole system-stored procedure（sp_helpdbfixedrole系统存储过程），493
sp_helpdevice system-stored procedure（sp_helpdevice系统存储过程），800

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

sp_helprole system-stored procedure（sp_helprole系统存储过程），495
sp_helprolemember system-stored procedure（sp_helprolemember系统存储过程），493
sp_helpserverrole system-stored procedure（sp_helpserverrole系统存储过程），487
sp_helpsrvrolemember system-stored procedure（sp_helpsrvrolemember系统存储过程），487
sp_helpuser system-stored procedure（sp_helpuser系统存储过程），490
sp_procoption system-stored procedure（sp_procoption系统存储过程），333
sp_refreshsqlmodule system-stored procedure（sp_refreshsqlmodule系统存储过程），244
sp_refreshview system-stored procedure（sp_refreshview系统存储过程），245
sp_rename system-stored procedure（sp_rename系统存储过程），687
sp_serveroption system-stored procedure（sp_serveroption系统存储过程），726
sp_setapprole stored procedure（sp_setapprole存储过程），497
sp_settriggerorder system-stored procedure（sp_settriggerorder系统存储过程），398
sp_spaceused system-stored procedure（sp_spaceused系统存储过程），663, 667
sp_updatestats system-stored procedure（sp_updatestats系统存储过程），760, 764
sp_who system-stored procedure（sp_who系统存储过程），134
SQL collations（SQL排序规则），155
SQL injection attacks（SQL注入攻击），326, 767
SQL instance-scoped events, capturing（捕获SQL实例范围的事件），533–535,539
SQL logins，accessing non-SQL server resources and（SQL登录名，访问非SQL Server资源），528
SQL Native Client (Microsoft) OLE DB Provider，724, 734
SQL Server 2000，XML integration and（XML集成），430
SQL Server 2008
 BULK option and（BULK选项），723, 735–738
 cascading changes and（级联修改），163
 Change Data Capture new with（变更数据捕获），93, 96–107
 Change Tracking new with（新的更改跟踪），93, 107–113
 CLR integration and（CLR集成），401
 CLR support，enabling in（CLR支持，启用），405
 compound assignment operators new with（新的组合赋值运算符），84–86
 Data Collector new with（新的数据收集器），748
 data compression improvements with（数据压缩的提升），192–195
 data page restores and（数据页存储），819
 data stream compression new with（新的数据流压缩），697, 711
 database mirroring and（数据库镜像），801
 database object dependency tracking and（数据库对象依赖关系跟踪），687,690–695
 DATE_CORRELATION_OPTIMIZATION option and（DATE_CORRELATION_OPTIMIZATION选项），643
 DDL triggers and（DDL触发器），373, 386
 FILESTREAM attribute new with（新的FILESTREAM属性），80–84
 filtered indexes new with（新的筛选索引），212, 214
 FORCESEEK hint new with（新的FORCESEEK提示），456–458
 forcing to use query execution plans and（强制使用查询执行计划），769
 grouping sets and（分组集），49
 hierarchyid data type new with（新的分层数据类型），435–441
 index compression new with（新的索引压缩），212
 index options and（索引选项），199，208–212
 join types and（联结类型），29
 loading assemblies into（加载程序集），409
 lock escalation and（锁升级），124, 126
 lock resources of（锁定资源），123
 MERGE command new with（新的MERGE命令），89, 90–93
 multiple rows from single INSERT statement new with（新的在单个INSERT语句中插入多个行），71
 object dependencies new with（新的对象依赖关系），244
 partial backups and（部分备份），807
 query guide creation from cache new with（新的从缓存创建查询指南），775–777
 query performance tuning and（查询性能调优），740
 read-consistency options and（读一致性选项），647–648
 rebuilding heaps new with（新的堆重建），669, 686
 Resource Governor feature new with（新的资源调配器），781–787
 Server Audit object new with（新的服务器审核对象），501, 529–545
 Service Broker and, 579–306
 sparse columns new with（新的稀疏列），150–153
 spatial data new with（新的空间数据），442–448
 statement-level stored procedure recompilation new with（新的语句级存储过程重编译），453
 sys.fn_validate_plan_guide system catalog function new with（新的sys.fn_validate_plan_guide系统目录函数），777
 sys.sql_expression_dependencies view new with（新的sys.sql_expression_dependencies视图），687
 table partitioning improvements with（分区表的提升），180, 189
 table-valued parameters new with（新的表值参数），369–372
 Transparent Data Encryption (TDE) new with（透明数据加密），547, 575–578
SQL Server data type to CLR data type translations（SQL Server数据类型到CLR数据类型的事务），407
SQL Server instances，shutting down（SQL实例，关闭），505
SQL Server logins（SQL Server登录名），480–485
SQL Server principals（SQL Server主体），475, 480–488

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

SQL Server Profiler，742
SQL Server transactional control（SQL Server事务控制），80–84
SQRT function（SQRT函数），262
SQUARE function（SQUARE函数），262
STArea method（STArea方法），445
statistical functions（统计函数），260,298
statistics（统计信息），760–766
 creating（创建），760
 filtered（筛选），762
 query performance and（查询性能），740
 updating（更新），762
 viewing detailed information about（查看详细信息），765
STATISTICS commands（STATISTICS命令），746–766
STDEV function（STDEV函数），258, 261
STDEVP function（STDEVP函数），258, 261
STDistance method（STDistance方法），445
STIntersects method（STIntersects方法），446
STLength method（STLength方法），445
stoplists/stopwords（非索引字列表/非索引字），217, 226–229
stored procedure headers（存储过程头信息），335
stored procedures（存储过程），325–342, 404
 best practices for（最佳实践），741
 creating（创建），326–330
 documenting（文档化），335
 dropping（删除），332
 executing automatically（自动执行），333
 extended（扩展），403
 for processing messages（处理消息），598–601
 inserting table data via（插入表数据），70
 modifying（修改），332
 TRY...CATCH command and（TRY...CATCH命令），466, 470
 when to use/benefits of（何时使用/益处），325
STR function（STR函数），264
string concatenation（字符串拼接），24
string execution（字符串执行），767
string functions（字符串函数），263–274
strings, discarding common from full-text indexes（丢弃全文索引中普通字符串），226–229
striping backups（条带化备份），799
STUFF function（STUFF函数），264, 270, 354
subqueries（子查询），27
SUBSTRING function（SUBSTRING函数），264, 273
subtract, assign (-=) operator（减等于，赋值运算符），84
SUM aggregate function（SUM聚合函数），257, 260
summarizing data（汇总数据），46–56
surrogate keys（代理键），165–567, 359
symmetric keys（对称密钥），547
SYNONYM object（SYNONYM对象），723
synonyms（同义词），732
sys.asymmetric_keys system catalog view（sys.asymmetric_keys系统目录视图），556
sys.backup_devices catalog view（sys.backup_devices目录视图），799
sys.certificates system catalog view（sys.certificates系统目录视图），568
sys.configurations system catalog view（sys.configurations系统目录视图），615
sys.conversation_priorities catalog view（sys.conversation_priorities目录视图），596
sys.databases system catalog view（sys.databases系统目录视图），636, 644, 650
sys.database_audit_specifications system catalog view（sys.database_audit_specifications系统目录视图），537
sys.database_audit_specification_details system catalog view（sys.database_audit_specification_details系统目录视图），537
sys.database_mirroring system catalog view（sys.database_mirroring系统目录视图），713–715
sys.database_mirroring_endpoints system catalog view（sys.database_mirroring_endpoints系统目录视图），703
sys.database_recovery_status catalog view（sys.database_recovery_status目录视图），820
sys.dm_audit_actions catalog view（sys.dm_audit_actions目录视图），533, 535, 541
sys.dm_database_encryption_keys catalog view（sys.dm_database_encryption_keys目录视图），577
sys.dm_db_index_physical_stats dynamic management view（sys.dm_db_index_physical_stats动态管理视图），756–758
sys.dm_db_index_usage_stats dynamic management view（sys.dm_db_index_usage_stats动态管理视图），759
sys.dm_exec_cached_plans dynamic management view（sys.dm_exec_cached_plans动态管理视图），778, 781
sys.dm_exec_cached_plans system catalog view（sys.dm_exec_cached_plans系统目录视图），644
sys.dm_exec_connections dynamic management view（sys.dm_exec_connections动态管理视图），121, 136
sys.dm_exec_query_plan dynamic management view（sys.dm_exec_query_plan动态管理视图），645, 769
sys.dm_exec_query_stats dynamic management view（sys.dm_exec_query_stats动态管理视图），749, 777
sys.dm_exec_requests dynamic management view（sys.dm_exec_requests动态管理视图），121, 742
sys.dm_exec_sql_text dynamic management view（sys.dm_exec_sql_text动态管理视图），121, 134, 136, 749
sys.dm_exec_text_query_plan dynamic management view（sys.dm_exec_text_query_plan动态管理视图），646
sys.dm_io_virtual_file_stats dynamic management view（sys.dm_io_virtual_file_stats动态管理视图），753
sys.dm_os_performance_counters dynamic management view（sys.dm_os_performance_counters动态管理视图），731
sys.dm_os_waiting_tasks dynamic management view（sys.dm_os_waiting_tasks动态管理视图），134
sys.dm_os_wait_stats dynamic management view（sys.dm_os_wait_stats动态管理视图），752
sys.dm_resource_governor_resource_pools dynamic management view（sys.dm_resource_governor_resource_pools动态管理视图），787

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

sys.dm_resource_governor_workload_groups dynamic management view（sys.dm_resource_governor_workload_groups动态管理视图），787
sys.dm_sql_referenced_entities dynamic management view（sys.dm_sql_referenced_entities动态管理视图），687, 692
sys.dm_sql_referencing_entities dynamic management view（sys.dm_sql_referencing_entities动态管理视图），687, 692
sys.dm_tran_active_transactions dynamic management view（sys.dm_tran_active_transactions动态管理视图），121
sys.dm_tran_session_transactions dynamic management view（sys.dm_tran_session_transactions动态管理视图），121
sys.endpoints catalog view（sys.endpoints目录视图），527
sys.fn_builtin_permissions system catalog function（sys.fn_builtin_permissions系统目录函数），504
sys.fn_validate_plan_guide system catalog function（sys.fn_validate_plan_guide系统目录函数），777
sys.linked_logins catalog view（sys.linked_logins目录视图），729
sys.messages system catalog view（sys.messages系统目录视图），459
sys.messages table（sys.messages表），459, 462
sys.plan_guides catalog view（sys.plan_guides目录视图），774, 777
sys.resource_governor_resource_pools catalog view（sys.resource_governor_resource_pools目录视图），784
sys.resource_governor_workload_groups catalog view（sys.resource_governor_workload_groups目录视图），785
sys.schemas system catalog view（sys.schemas系统目录视图），518
sys.servers system catalog view（sys.servers系统目录视图），727, 729
sys.server_audit_specifications catalog view（系统目录视图），534
sys.server_audits catalog view（sys.server_audits目录视图），531
sys.server_audit_specification_details catalog view（sys.server_audit_specification_details目录视图），534
sys.server_file_audits catalog view（sys.server_file_audits目录视图），531
sys.server_permissions catalog view（sys.server_permissions目录视图），508
sys.server_principals system catalog view（sys.server_principals系统目录视图），477, 482,486, 508, 527, 729
sys.services catalog view（sys.services目录视图），597
sys.service_contracts catalog view（sys.service_contracts目录视图），597
sys.sql_dependencies catalog view（sys.sql_dependencies目录视图），690
sys.sql_expression_dependencies catalog view（sys.sql_expression_dependencies目录视图），687, 690
sys.sql_modules catalog view（sys.sql_modules目录视图），242, 325, 334
sys.symmetric_keys system catalog view（sys.symmetric_keys系统目录视图），562
sys.triggers catalog view（sys.triggers目录视图），385
sys.XML_schema_collections catalog view（sys.XML_schema_collections目录视图），423
sys.XML_schema_namespaces catalog view（sys.XML_schema_namespaces目录视图），423
sysadmin fixed server role（sysadmin固定服务器角色），485
sysadmin server role members（sysadmin服务器角色成员），549
SYSDATETIME function（SYSDATETIME函数），278
SYSDATETIMEOFFSET function（SYSDATETIMEOFFSET函数），144, 278
system catalog views（系统目录视图），225
system-defined error messages（系统定义错误消息），459–462
system functions（系统函数），293–303
SYSTEM_USER function（SYSTEM_USER函数），302
SYSUTCDATETIME function（SYSUTCDATETIME函数），278

T

T-SQL programming language（T-SQL编程语言），401
table aliases（表别名），29, 32
table constraints（表约束），157
table hints（表提示），454
table lock resource（表锁资源），124
table partitioning（表分区），180, 181–191
table-valued functions，invoking for outerquery rows（表值函数，外部查询行时调用），35–38
table-valued parameters（表值参数），369–372
table-valued user-defined functions（表值用户定义函数），343, 357
table variables（表变量），176–180
 performance issues and（性能问题），177
 RETURNS keyword and（RETURNS关键字），352
tables（表）
 creating（创建），147
 derived（衍生），33
 dropping（删除），154
 filegroups，placing on（文件组，放在），180, 191
 new，creating from query results via INTO clause（新的，通过INTO子句从查询结果创建），26
 querying multiple（多表查询），28–35
 referencing multiple times（多次引用），32
 renaming（重命名），687–689
 reporting information about（报告信息），154
 statistics，generating/updating across（统计信息，生成/更新），763–764
 temporary（临时），176–180
 truncating（截断），88
 validating integrity of（验证完整性），674–682
 very large，managing（超大型，管理），180–578
tempdb database（tempdb数据库），176, 207
temporary tables（临时表），176–180
 vs. derived tables（相对于衍生表），33
 performance issues and（性能问题），177
term proximity（词的相邻），217, 234
tessellation process（分割过程），448

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

testing，best practices for（测试，最佳实践），741
text data types (deprecated with SQL Server 2005)（text数据类型，SQL Server 2005不建议使用），76
time（时间），277, 285
@@TIMETICKS function（@@TIMETICKS函数），298
TODATETIMEOFFSET function（TODATETIMEOFFSET函数），278
tokens（标记），217
TOP keyword（TOP关键字），19, 89
TORN_PAGE_DETECTION option（TORN_PAGE_DETECTION选项），650
ToString method（ToString方法），437
@@TOTAL_ERRORS function（@@TOTAL_ERRORS函数），298
@@TOTAL_READ function（@@TOTAL_READ函数），298
@@TOTAL_WRITE function（@@TOTAL_WRITE函数），298
trace flags（跟踪标志），138–141
Transact-SQL cursors，temporary tables as alternative to（T-SQL游标，作为备份方案的临时表），176
Transact-SQL predicates（T-SQL谓词），230–232
Transact-SQL scripts（T-SQL脚本），23
Transact-SQL statements（T-SQL语句）
 transaction locking behavior for（事务锁定行为），129–133
 default transaction locking behavior for(默认事务锁定行为），129–133
transaction log（事务日志）
 backing up（备份）698, 700, 705–707, 791, 803
 recovery models for（恢复模式），649
 relocating（重定位），657
 restoring backups of（还原备份），812–815
 removing from database（从数据库中移除），656
transaction mark recovery（事务标记还原），649
transactions（事务），115–122
 concurrent，interactions between（并发，交互），128
 configuring locking behavior for（配置锁行为），129–133
 displaying oldest active（显示最早的活动），119
 DML triggers and（DML触发器），381
 obtaining information about（获得信息），120
 recommendations for handling in Transact-SQL code（在T-SQL代码中处理的建议），119
 types of（类型），115
@@TRANCOUNT function（@@TRANCOUNT函数），297
@@TRANCOUNT TRANSACTION command（@@TRANCOUNT TRANSACTION命令），117
Transparent Data Encryption (TDE)（透明数据加密），547, 575–578
transport security,（传输安全）601, 603–606
trigger recursion（触发器递归），396
triggers（触发器），373–400, 404
 CLR，creating（CLR，创建），415–416
 dropping（删除），399
 enabling/disabling（启用/禁用），394
 firing order and（触发顺序），397
 INSTEAD OF, 247
 logon（登录），390
 managing（管理），393–400
 TRY...CATCH command and（TRY...CATCH命令），466
troubleshooting（故障处理）
 active transactions（活动的事务）120
 blocks（块），134–137
 deadlocks（死锁），137–142
 foreign keys（外键），163
TRUNCATE TABLE command（TRUNCATE TABLE命令），88，337
TRUSTWORTHY option（TRUSTWORTHY选项），581
TRY...CATCH command（TRY...CATCH命令），465, 468–473
 benefits of（益处），465
 nesting calls and（嵌套），471
 stored procedures and（存储过程），470

U

UDFs。参见user-defined functions
UDTs。参见user-defined types
undo phase（重做阶段），810
Unicode files，importing（Unicode文件，导入），735–738
UNICODE function（UNICODE函数），264
UNION ALL operator（UNION ALL运算符），35, 49, 51
UNION ALL statement（UNION ALL语句），239, 251, 255
UNION operator，result sets and（UNION运算符，结果集），33
UNION statement（UNION语句），364
UNIQUE constraints（UNIQUE约束），168–171
unique indexes（唯一索引），199
uniqueidentifier column（唯一标识符列），67
uniqueidentifier data type（唯一标识符数据类型），165, 303–306, 589
UNKNOWN result（UNKNOWN结果），7
UNPIVOT command（UNPIVOT命令），42
UNSAFE permission（UNSAFE权限），406，410
untyped XML columns（无类型XML列），422
update locks（更新锁），123
update mask（更新掩码），104
UPDATE statement（UPDATE语句）73–85
 MERGE command and（MERGE命令），90–93
 OUTPUT clause, using with（使用OUTPUT子句），93
UPDATE STATISTICS command（UPDATE STATISTICS命令），762
updates，lost（更新，丢失），128
UPDLOCK hint（UPDLOCK提示），455
UPPER function（UPPER函数），264, 270
USE clause（USE子句），2
USE PLAN 'xml_plan' hint（USE PLAN 'xml_plan'提示），452
USE PLAN command（USE PLAN命令），771, 769
user access modes（用户访问模式），628
user-defined aggregate functions（用户定义聚合函数），404
user-defined database roles（用户定义数据库角色），488, 495–497
user-defined error messages（用户定义错误消息），459–462

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

user-defined functions (UDFs)（用户定义函数），343–365, 404
benefits of（益处），357–365
cautions for（警告），357
dropping（删除），356
modifying（修改），354
table-valued parameters and（表值参数），369–372
viewing list of（查看列表），356
user-defined types(UDTs)(用户定义类型)，343, 365–372, 404
caution for（警告），365
creating（创建），365–367
dropping（删除），368
USER function（USER函数）302
UTC Coordinated (Universal Time)（UTC（协调世界时）），144, 278
utilities（实用程序）
Database Engine Tuning Advisor（数据库引擎调优顾问），755
SQL Server Profiler，742

V

VALUE indexes（VALUE索引），429
values（值）
assigning to variables（为变量赋值），12
converting into columns（转化为列），39
declaring/assigning values to（声明/赋值），12
duplicate，removing（复制，删除），21
inserting explicitly into IDENTITY column（显式地插入到IDENTITY列），66
NULL，checking for（NULL，检查），10
returning rows based on list of（返回基于列表的行），11
VALUES keyword（VALUES关键字）71–73
VAR function（VAR函数），258, 260
varbinary(max) data types（varbinary(max)数据类型），76
varchar(max) data types（varchar(max)数据类型），76
VARP function（VARP函数），258, 260
VB.NET，creating assemblies in（VB.NET，创建程序集），405
vbc.exe compiler（vbc.exe编译器），408
very large databases (VLDBs)，filegroups and（超大型数据库（VLDB），文件组），180,191
view definitions（查看定义）
querying（查询），242
refreshing（刷新），244
viewing（查看）
aggregated performance statistics（聚合的性能统计信息），750
asymmetric keys（非对称密钥），556
backup devices（备份设备），800
cached query plans，statistics for(缓存查询计划，统计信息），748
database certificates（数据库证书），568
database information（数据库信息），622–624
database object definitions（数据库对象定义），694
database options（数据库选项），636
database space usage（数据库空间使用），663–665
execution runtime information（执行运行时信息），746–748
fixed database role members（固定数据库角色成员），493
fixed database roles（固定数据库角色），493
fixed server role members（固定服务器角色成员），487
fixed server roles（固定服务器角色），486
index usage（索引使用），754, 759–760
indexes（索引），203
linked logins（链接登录名），729
linked servers（链接服务器），727
media sets，contents of（媒体集，内容），808–809
permissions，available（权限，可用的），503
query execution plans（查询执行计划），743–771
SQL Server configurations（SQL Server配置），615–617
SQL Server logins（SQL Server登录名），482
statistics information（统计信息），765
symmetric keys（对称密钥），562
system catalog views and（系统目录视图），225
Windows logins（Windows登录名），477
views（视图），239–256
CTEs and, 56–62
dropping（删除），245
encrypting（加密），247
indexed（索引），248–251
partitioned（分区），251–256
regular（常规的），240–247
replacing with multi-statement UDFs（用多语句UDF替代），362–365
returning information about（返回信息），243
virtual log files (VLFs)（虚拟日志文件VLF），668
VLDBs (very large databases)，filegroups and（VLDB（超大型数据库），文件组），180，191
VLFs (virtual log files)（VLF（虚拟日志文件）），668

W

WAITFOR command（WAITFOR命令），307, 313, 318
waits（等待），752
Waits and Queues methodology（等待和队列方法论），752
weighted value，returning search results by（加权值，返回搜索结果），236
Well-Known Binary (WKB)（熟知二进制），442
Well-Known Text (WKT)（熟知二文本），442
WHERE command（WHERE命令），3–7
keeping unambiguous（保持无歧义），6
rows，updating via（行，更新），75
WHERE statement（语句），307, 312, 314
best practices for（最佳实践），741
predicates and（谓词），230
temporary tables and（临时表），176
wildcard searches（通配符搜索），217, 233
wildcards，with LIKE（通配符，使用LIKE），11
Windows authentication，Windows principals and（Windows身份验证，Windows主体），475–480
Windows collations（Windows排序规则），155
Windows credentials（Windows凭据），528

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Windows logins（Windows登录名），475–480
creating（创建），476
dropping（删除），479
Windows-only authentication（只用Windows身份验证），476
Windows principals（Windows主体），475–480
WITH CUBE clause（WITH CUBE子句），46
WITH NORECOVERY option，database mirroring and（WITH NORECOVERY选项，数据库镜像），700
WITH RECOMPILE command（WITH RECOMPILE命令），340
WITH ROLLUP clause（WITH ROLLUP子句），49
witness server（见证服务器），699
WKB (Well-Known Binary)（熟知二进制），442
WKT (Well-Known Text)（熟知文本），442
word meaning，returning search results by（单词含义，返回搜索结果），235
workload groups（工作负荷组），782
worktables（工作表），748

X

XLOCK hint（XLOCK提示），455
xml data type（xml数据类型），365, 419–430
XML data，retrieving/modifying（XML数据，检索/修改），424–428
XML DML，428
XML format, converting relational data sets into（将相关的数据集转化为XML格式），430–435
XML indexes（XML索引），428–430
XML integration（XML集成），419–435
XML Schema collections（XML架构排序规则），422–424
XML transport formats（XML传输格式），442
XQuery methods（XQuery方法），424–428

Y

YEAR function（YEAR函数），278, 283

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。